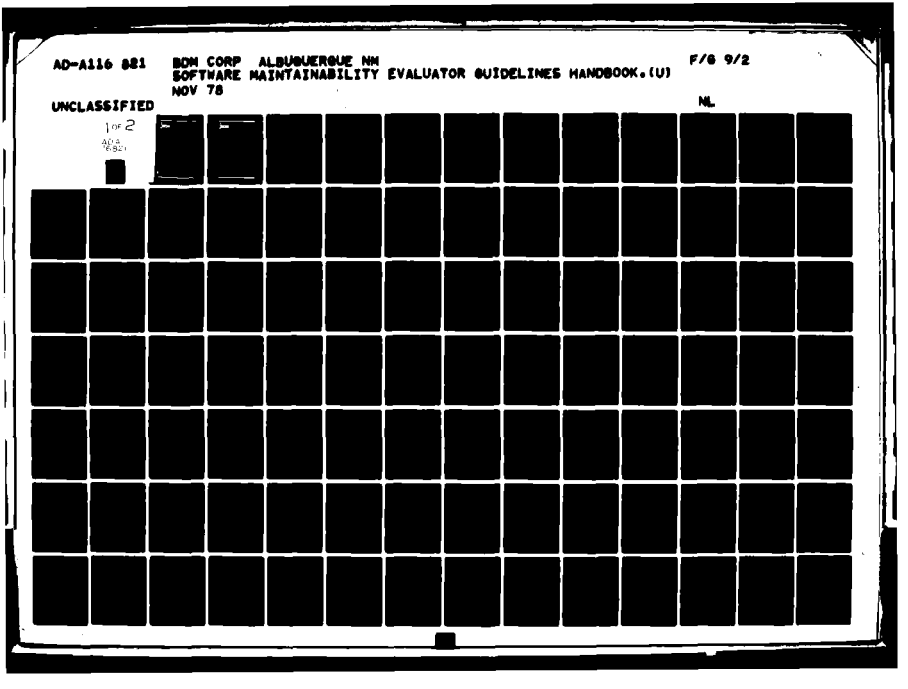
AD-A116 881 BDM CORP ALBUQUERQUE NM F/G 9/2
SOFTWARE MAINTAINABILITY EVALUATOR GUIDELINES HANDBOOK.(U)
NOV 78

UNCLASSIFIED NL

1 OF 2
AD-A116 881



THE BDM CORPORATION

901 RANDOLPH ROAD, S.E. · ALBUQUERQUE, NEW MEXICO 87106 · (505) 848-5000 · TWX 910-989-0619

THE BDM CORPORATION

1801 RANDOLPH ROAD, S.E. · ALBUQUERQUE, NEW MEXICO 87106 · (505) 848-5000 · TWX 910-989-0619

SOFTWARE MAINTAINABILITY EVALUATOR GUIDELINES HANDBOOK

24 November 1978

BDM/TAC-78-687-TR

PREFACE

This Software Maintainability Evaluator Guidelines Handbook provides the information which an evaluator will need to understand and complete an evaluation of the maintainability of a software program. The evaluation methodology will continue to evolve and the evaluator should understand that this type of evaluation will never be complete in itself. It should serve as a tool to identify possible problem areas in maintaining a software program and as a complementary part of the total software evaluation process.

This handbook is separated into four sections. Section I contains the general Software Maintainability Evaluation Methodology including definitions and the major hierarchical organization. Section II contains specific evaluator guidelines for entering answers on questionnaire response forms. Examples of completed response forms are included. Section III contains a copy of the Software Documentation Questionnaire and the Module Source Listing Questionnaire. Section IV provides the evaluator with a brief discussion of each question by questionnaire as well as examples to illustrate some of the concepts addressed by the evaluation.

It is recommended that the evaluator completely scan this handbook to become familiar with its format and contents. Section II should be read in detail in order that the specific evaluation procedure and response requirements are understood. Sections I and IV are primarily intended to be supplementary to the evaluation. However, the evaluator should be aware that specific response requirements on a question basis are given in Section IV and some general evaluation questions require the evaluator to understand the definitions as presented in Section I.

In addition to this handbook, the evaluator will be supplied with the appropriate response forms and the specific program-related identification information as explained in Section II of this handbook.

TABLE OF CONTENTS

SOFTWARE MAINTAINABILITY
EVALUATOR GUIDELINE HANDBOOK

LIST OF FIGURES

LIST OF TABLES

SECTION I
SOFTWARE MAINTAINABILITY EVALUATION METHODOLOGY

A. GENERAL METHODOLOGY

Software consists of the programs and documentation which result from a software development process. The manner in which the software is physically grouped (e.g., order of development, functional purpose, etc.) determines what are called the software products. The terminology for what the products are called or of what the products consist have some general agreement but no rigorous definitions.

In order to evaluate software in a systematic manner it is important to understand what quality or qualities are most important from a user's viewpoint. And, there must be a method of associating the generally understood software qualities to more specific and measureable software characteristics which become the basis for the evaluation.

1. Quality Factors

Software qualities at the user level are termed quality factors. Examples of such quality factors might include maintainability, reliability, efficiency, correctness, usability, portability, human engineering, etc. Given a quality factor to evaluate it is necessary to relate that quality to specific software characteristics each of which can be measured using the proposed measurement tool. In order to bridge the rather large gap between user understanding and the software itself, an intermediate set of user-oriented software qualities termed test factors or criteria are defined. These test factors should be relatively non-overlapping and, as a group, representative of the given quality factor. Individual characteristics of the software products to be evaluated can now be grouped under the most appropriate software test factor. The hierarchy as discussed above is illustrated by figure I-1.

The individual metric (score) for each characteristic is accumulated (via some function) to obtain the test factor or criterion metric and each test factor metric is accumulated (via some function) to obtain the overall metric for the quality factor. The generation of metrics at various levels (quality factor, test factor, characteristic) helps to focus on where potential problem areas might exist.

The actual determination of generic factors, characteristics, measurement tools, software products is not a simple task. And, since the usual purpose is to produce a generic evaluation approach applicable to a large (if not all) class of software, the task will certainly involve an understanding of software state-of-the-art.

2. Software Product Activity

In order to better scope the particular software evaluation, it is important to understand what software products are being evaluated and what general product activities are being considered. A list of typical software documentation products is given in table I-1. The primary areas of product activity with which various software quality factors would be associated include product operability, product revision, and product transportability. Depending upon the particular combination of product activities which are of interest and which particular quality factors within each activity area are most important for a given application, an overall set of requirements for software quality can be generated. As an example, perhaps "Usability" from product operation, "Maintainability" from product revision and "Portability" from product transportability are of particular importance.

It should also be noted that not all software qualities are complementary. For example, the very characteristics which in a given application have a positive effect on maintainability may well have a negative effect on efficiency. This fact is due in part to the current state-of-the-art in hardware architecture versus software architecture in that there are usually no directly efficient implementations of high order language constructs. Thus, there is the commonly held view that assembly language is the necessary source language for real-time programs which require high efficiency. The evaluation of software for

one quality in general will not reflect the presence or absence of any other qualities except where there are some common characteristics which cause either strong positive or negative correlation.

3. Software Evaluation Procedure

The actual evaluation procedure may take one of three basic forms: objective via automated tools, subjective via questionnaires and evaluators, or a combination of the objective and subjective forms. The evaluation procedure described here is primarily subjective. A set of evaluators completes closed form (fixed number of responses per question) questionnaires (with capability for optional written comments) designed to determine the presence or absence of certain quality characteristics in a given software product or products. The scope of the questionnaire depends upon the organization of the products to be evaluated and the particular quality factors and associated test factors for which the products are evaluated.

B. MAINTAINABILITY EVALUATION METHODOLOGY

The software quality for which this evaluation is designed is maintainability. In line with the general methodological approach outlined in Section I.A the software products and maintainability test factors have been identified for the maintainability evaluation. In addition, the specific evaluation measurement tool (closed form questionnaires) and a detailed procedure for assuring reasonably reliable results have been developed. The products, test factors, evaluation tools and procedures will be briefly discussed in the next few subsections. A set of definitions which will be used for the maintainability evaluation is summarized in table I-2. The Elements of Software Maintainability are illustrated in figure I-3. This hiearchical evaluation structure enables one to identify potential maintainability problems at various levels: product (documentation, module source listings), quality (maintainability, test factors), combinations of product and quality (e.g., module A modularity, or documentation descriptiveness, or module B overall maintainability, etc.).

1. Software Products

Software consists of a set of computer (software) programs and the associated documentation on the design, implementation, test, support and operational procedures of those programs. Figure I-2 illustrates a computer system hierarchy as it applies to a maintainability evaluation. Each software program is separately evaluated and consists of a set of components called modules. A module may in general be at any conceptual level of the program. The highest level is the program itself and the lowest (more usual) level is the smallest separately "callable" unit of contiguous code or perhaps a small collection of such units of code. Within this hierarchy there are three related products which are evaluated for characteristics which affect the maintainability of a software program. The products are the software program's documentation, the software program's source listings and the computer support resource's documentation. It is important to emphasize that only deliverable products are to be considered in an evaluation.

a. Software Documentation

Software program documentation is the set of requirements, design specifications, guidelines, operational procedures, test information, problem reports, etc. which in total form the written description of a software program. The primary documentation which is used in the program evaluation consist of the documents containing program design specifications, program test plan information and procedures, and program maintenance information. These documents may have a variety of physical organizations depending upon the particular application. The documents are evaluated both for content and for general physical structure (format). The content evaluation is primarily concerned with how well the overall program has been designed (as documented) for maintainability. The format evaluation is primarily aimed at how the physical structure of the documentation (table of contents, index, numbering schemes, modular separation of parts, etc.) aids in understanding or locating program information as might be done during program maintenance.

b. Software Source Listings

Software source listings are the computer generated (or equivalent) form of the program code in its source language (e.g., FORTRAN, COBOL, PL/I, PASCAL, JOVIAL, ASSEMBLER, etc.). The source listings represent the program as implemented, in contrast to the documentation which for the most part represents the program design or implementation plan. The source listings for a given program will include listings for each separately specified module as well as the available support information such as cross reference listings. Source listings are often bound and delivered as a document from the contractor. In essence, source listings are also a form of program documentation, but for this maintainability evaluation, a distinction is made.

The source listing evaluation consists of a separate evaluation of each specified module's source listing and the consistency between the module's source listing and the related written module documentation. The separate module evaluations are then accumulated to give the overall evaluation of the software source listing for the given program. This hierarchical division allows for potential problems in maintainability to be identified at various levels.

c. Computer Support Resources

Computer support resources include all the relevant resources such as software, computer equipment, environment facilities (building), etc. which will be used to support the maintenance of the software being evaluated. The actual product which is evaluated as part of the maintainability evaluation is the documentation which should be available to explain the use and to some extent specification details of the support resources.

At this time, the appropriate hierarchy of products, test factors, and specific product questionnaires is in development and hence will not be further considered other than indirectly.

2. Software Maintainability Test Factors

The maintainability of software from an evaluation of at least the software products of documentation and source listings can be thought of as a function of six attributes or test factors: modularity, descriptiveness, consistency, simplicity, expandability and instrumentation. Definitions of maintainability and the six test factors, and discussions of their application in the evaluation of the documentation and source listings software products will be briefly given in the next few subsections. Figure I-3 illustrates the maintainability evaluation hierarchy. Figure I-4 illustrates the general framework of a product questionnaire with the modular grouping of the questions on individual product maintainability characteristics by test factor.

a. Maintainability

Software maintainability is a quality of software which reflects the effort required to perform the following actions:

(1) removal/correction of latent errors

(2) addition of new features/capabilities

(3) deletion of unused/undesirable features

(4) modification of software to be compatible with hardware changes

Implicit in the above definition are the concepts that software should be understandable in order to be able to identify the source of errors/change, modifiable in order to implement corrections/changes which have been identified, and testable in order to check out corrections/changes which have been implemented.

Software possesses the characteristics of understandability to the extent its purpose and organization are clear to the inspector.

Software possesses the characteristics of modifiability to the extent that it facilitates the incorporation of changes once the nature of the desired changes has been identified.

Software possesses the characteristics of testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.

b. Modularity

Software possesses the characteristics of modularity to the extent a logical partitioning of software into parts, components, modules has occurred.

Each software product is evaluated for modularity. A paraphrase of the definition is the extent to which the logical parts, components, modules are independent or show low coupling. It has been observed that software programs that have been the easiest to understand and change have been composed of independent modules.

The fewer and simpler the connections between parts, the easier it is to understand each module without reference to other parts. Minimizing connections between parts also minimizes the paths along which changes and errors can propagate into other parts of the system, thus eliminating disastrous "ripple" effects, where changes in one part cause errors in another, necessitating additional changes elsewhere, giving rise to new errors, etc. The widely used technique of using common program data areas (or global variables or modules without their own distinct set of variable names) can result in an enormous number of connections between the modules of a program. The modularity of a product is affected not only by the number of connections but by the degree to which each connection couples (associates) two parts, making them interdependent rather than independent. Coupling is the measure of the strength of association established by a connection from one part to another. Strong coupling complicates understanding and changeability.

Coupling between program modules increases with increasing complexity or obscurity of the interface. Coupling is lower when the connection is to the normal module interface than when the connection is to an internal component. Coupling is lower with data connections than with control connections, which are in turn lower than hybrid connections (modification of one module's code by another module). Control coupling, where a called module "tells" its caller what to do, is a more severe from of coupling.

When two or more program modules interface with the same area of storage, data region, or device, they share a common environment. Examples of common environment are:

(1) A set of data elements with the EXTERNAL attribute that is copied into PL/I modules via an INCLUDE statement or that is found listed in each of a number of modules.
(2) Data elements defined in COMMON statements in FORTRAN modules.
(3) A centrally located "control block" or set of control blocks.
(4) A common overlay region of memory.
(5) Global variable names defined over an entire program or section.

The most important structural characteristic of a common environment is that it couples every module sharing it to every other such module without regard to their functional relationship or its absence. For example, only the two modules XVECTOR and VELOC might actually make use of data element X in an "included" common environment of PL/I, yet changing the length of X impacts every module making any use of the common environment, and thus necessitates recompilation.

Coupling is reduced when the relationships among elements not in the same part are minimized. There are two ways of achieving this--minimizing the relationships among parts and maximizing relationships among elements in the same part. In practice, both ways are used.

As general guidelines it is important from a modularity viewpoint that modules consist of only a few easily recognizable functions which are closely related and that there are a minimal number of links to other modules - preferably only via parameters passed in a calling parameter list. In addition, the physical format of the documentation should exhibit component strength and lack of component coupling for its sections, volumes, etc. There should be separate sections for the description of the major parts which a given document's purpose encompasses. For example, the program specification document should include separate "sections" for the description of program interfaces, program major functions, etc. and these sections should be essentially self-contained (few cross references to other sections).

c. Descriptiveness

Software possesses the characteristics of descriptiveness to the extent that it contains information regarding its objectives, assumptions, inputs, processing, outputs, components, revision status, etc.

This quality is very important in being able to understand software. It is important that the documentation have a descriptive format and contain useful explanations of the software program design. The objectives, assumptions, inputs, etc. are useful at least in varying degrees of detail in both documentation and source listings. In addition, the intrinsic descriptiveness of the source language syntax and the judicious use of source commentary greatly aids efforts to understand the program operation.

d. Consistency

Software possesses the characteristics of consistency to the extent the software products correlate and contain uniform notation, terminology and symbology.

The use of some standards in documentation, flow chart construction and certain conventions in I/O processing, error processing, module interfacing, naming of modules/variables, etc. are typical reflections of consistency. Attention to consistency characteristics can greatly aid the ease of understanding the program. Consistency allows one to generalize easily. For example, programs using consistent conventions might require that the format of modules be similar. Thus by learning the format of one module (preface block, declaration format, error checks, etc.) the format of all modules is learned. This allows one to concentrate on understanding the true difficulties of an algorithm, data structure, etc.

e. Simplicity

Software possesses the characteristics of simplicity to the extent that it lacks complexity in organization, language, and implementation techniques and reflects the use of singularity concepts and fundamental structures.

The concept of software complexity (or lack of simplicity) can be very broadly interpreted. The aspects that are emphasized in the evaluation relate primarily to the concepts of size and primitives. The less there is to discriminate and the more use there is of basic or primitive techniques, structures, etc. the simpler the software will tend to be. The use of a high order language as opposed to an assembly language tends to make a program simpler to understand because there are fewer discriminations which have to be made. There are certain programming considerations such as dynamic allocation of resources, recursive/reentrant coding which can greatly complicate the data and control flow. Real-time programs, because of the requirement for timing constraints and efficiency tend to have more control complexity. The sheer bulk of counts (number of operators, operands, nested control structures, nested data structures, executable statements, statement labels, decision parameters, etc.) will determine to a great extent how simple or complex the source code is. The particular application itself may preclude the possibility of a reasonably simple design or implementation because of requirements such as a particularly complex real-time scheduling algorithm or high level mathematical or other theoretical considerations.

f. Expandability

Software possesses the characteristics of expandability to the extent that a physical change to information, computational functions, data storage or execution time can be easily accomplished.

Implicit in this definition is the concept that software is expandable if a change (insertion/deletion/modification) can be made once the nature of what is to be changed is understood. Software may be perfectly understandable but not easily expandable. If the design of the program has not allowed for a flexible timing scheme or a reasonable storage margin, then changes (even minor ones) may be extremely difficult to implement. Parameterization of constants and basic data structure sizes is helpful. If storage is available and a need to use a larger data structure (e.g., array) is required, then a simple change to the parameterized size (if dependencies on the structure size have

always utilized the parameter) completes the required modification. It is very important that the documentation include explanations of how to effect increases/decreases in basic array sizes or changes to the timing scheme. Or, at least the limitations of such program expandability should be clear. Usually the quality of expandability is either designed into the program from the beginning or it is not present in the implementation. Even the numbering schemes for documentation narrative and graphic materials must be carefully considered so that physical modifications to the documentation can be easily accomplished when necessary.

g. Instrumentation

Software possesses the characteristics of instrumentation to the extent it contains aids which enhance testing.

For the most part the documentation is evaluated on how well the program has been designed to include test aids (instruments), while the source listings are evaluated on how well the code seems to be implemented to allow for testing through the use of such test aids. And it is highly unlikely that a reasonably useful set of aids (debug tools, trace capability, data dump modules, test flags, test probes, performance monitors, test drivers, run time parameter checks, built-in error checks, input data range/type checks, etc.) will be available unless designed as part of the software from the beginning program design phase. This includes the design of the program test plan.

Because of the particular application (limited storage/timing flexibility), the software development environment (debug tools, dumps, etc.), the particular language chosen (support tools are not available and are too difficult to develop), or simply a lack of required emphasis on maintainability, contractors do not spend much design, implementation or integration time in instrumenting the software program. This part of the evaluation reflects the concern (from a maintainability viewpoint) that the software be designed and implemented so that instrumentation is imbedded within the program, can be easily inserted into the program (e.g., via dynamic diagnostic test probe capability), is available through a support software system, or perhaps is a combination thereof.

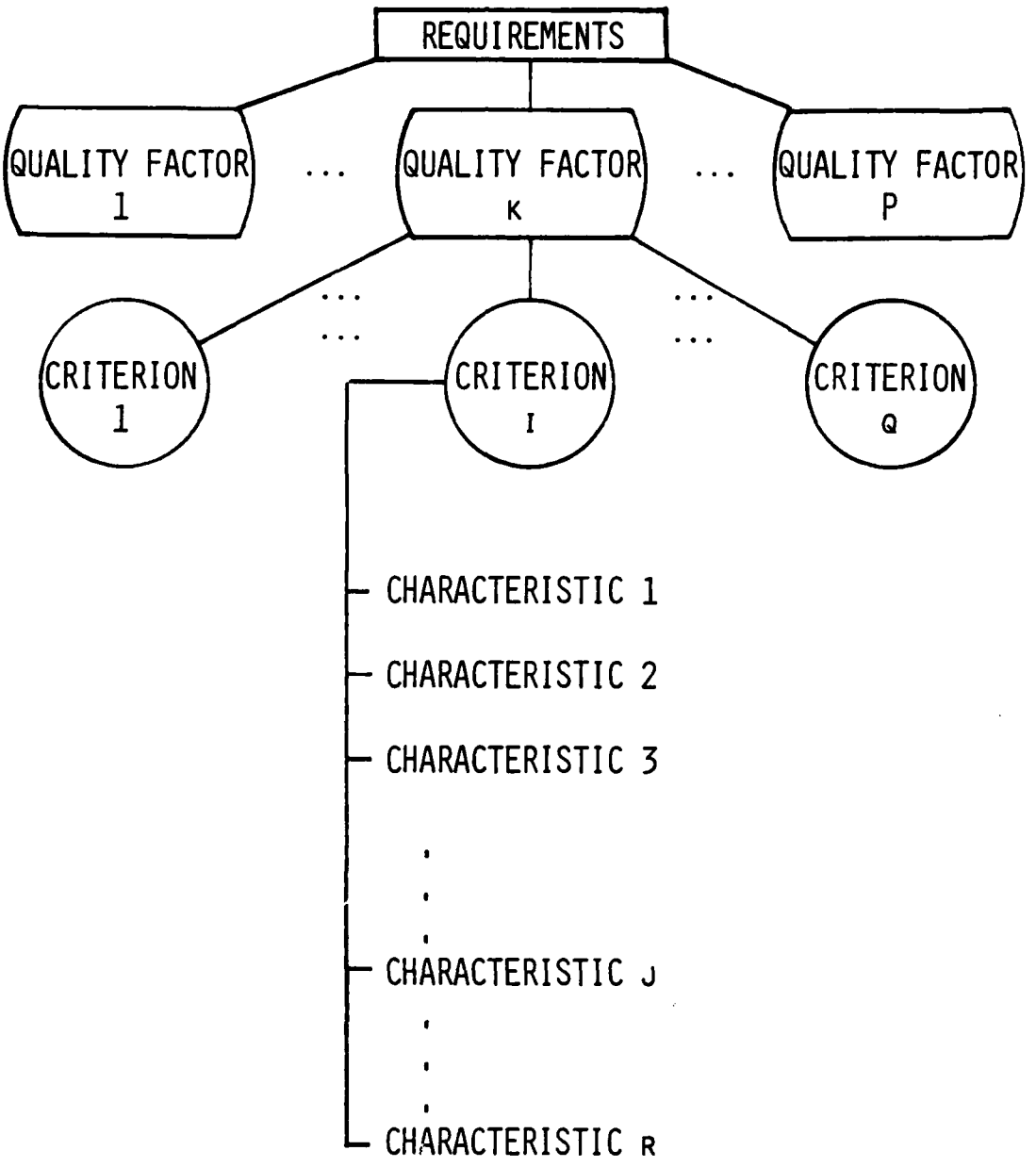
3. Software Maintainability Evaluation Procedure

The basic procedure for accomplishing a software maintainability evaluation is shown in figure I-4. Each software program is separately evaluated.

The program evaluation procedure is for each of a set of at least five evaluators knowledgeable in software maintenance to complete one documentation questionnaire, and a set of module source listing questionnaires, one on each of a set of randomly selected program modules. The responses are recorded on optical scanner answer sheets and processed through an automated optical scanner/maintainability analysis system. The reports output from this system indicate average and weighted scores across various levels (product, quality factor, test factor, by evaluator, specified groupings, etc.) as well as some statistics which can be analyzed to help assess how reliable the evaluation data itself might be. The scores can be used as indicators of potential problems in program maintenance, some of which may be correctable and others because of the nature of the application may be inherently uncorrectable.

In order to obtain a more reliable evaluation a clear understanding of the questions on each questionnaire and the specific question response guidelines is important. The function of the Pre-evaluation review briefing of evaluators and the Calibration Test is to accomplish this. The Pre-evaluation briefing of evaluators is optional since the guidelines are reasonably well-defined in this manual. In addition to the evaluation questionnaires, each evaluator will also complete a short questionnaire concerning his background and experience in software. This biodemographic questionnaire is used to determine whether any correlations exist (especially across many program evaluations) between any biodemographic statistic and the evaluator responses.

At the conclusion of the evaluation analysis, a summary of the software maintainability evaluation results and recommended action, if any, is included as part of the overall software program evaluation report (which includes other evaluation results besides maintainability).



$QF(K) = F_K(M_1, M_2, \ldots, M_Q)$ = QUALITY FACTOR RATING

WHERE M_I = MEASURE (METRIC VALUE) FOR CRITERION I

Figure I-1. Software Quality Evaluation Hierarchy

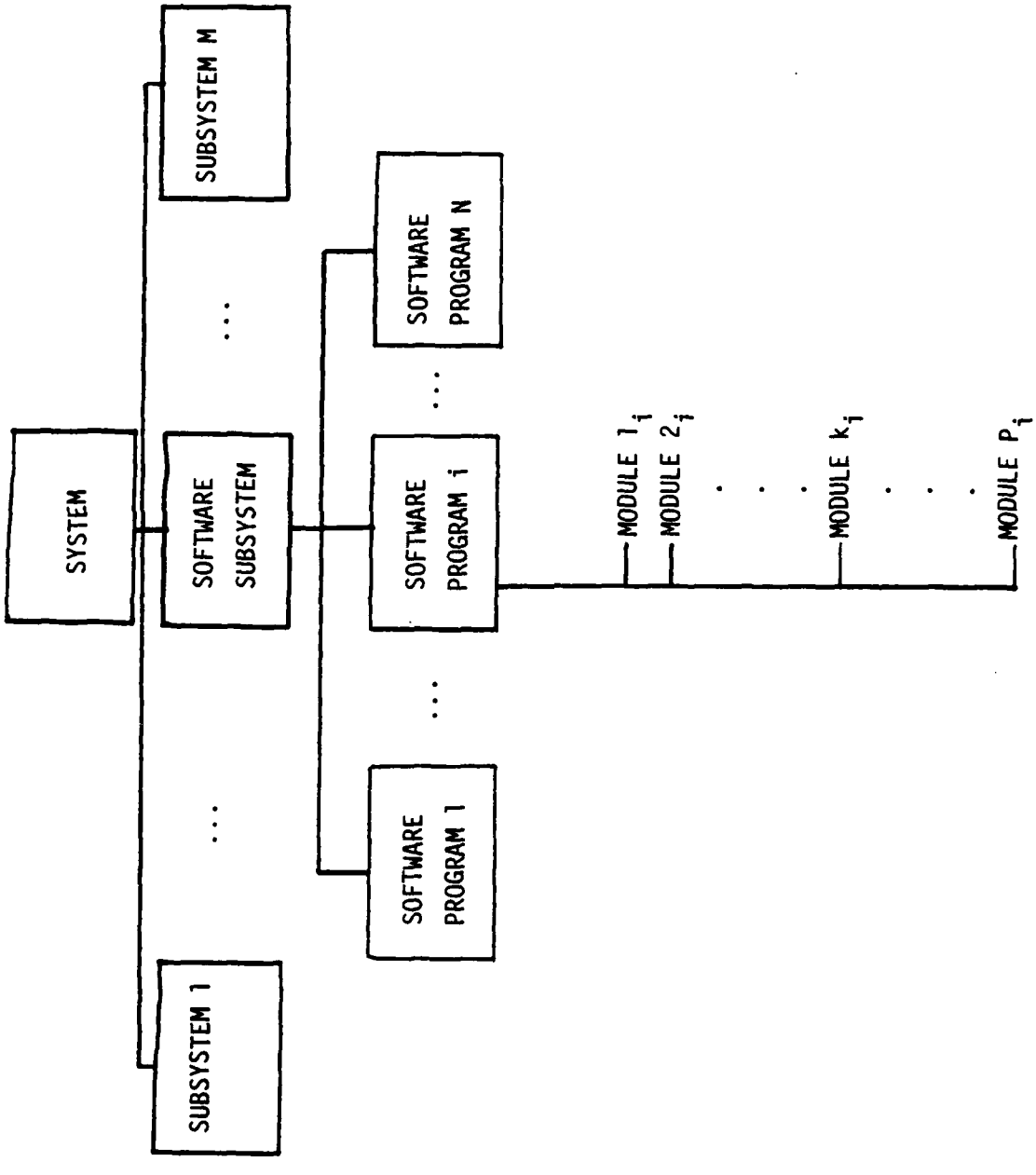
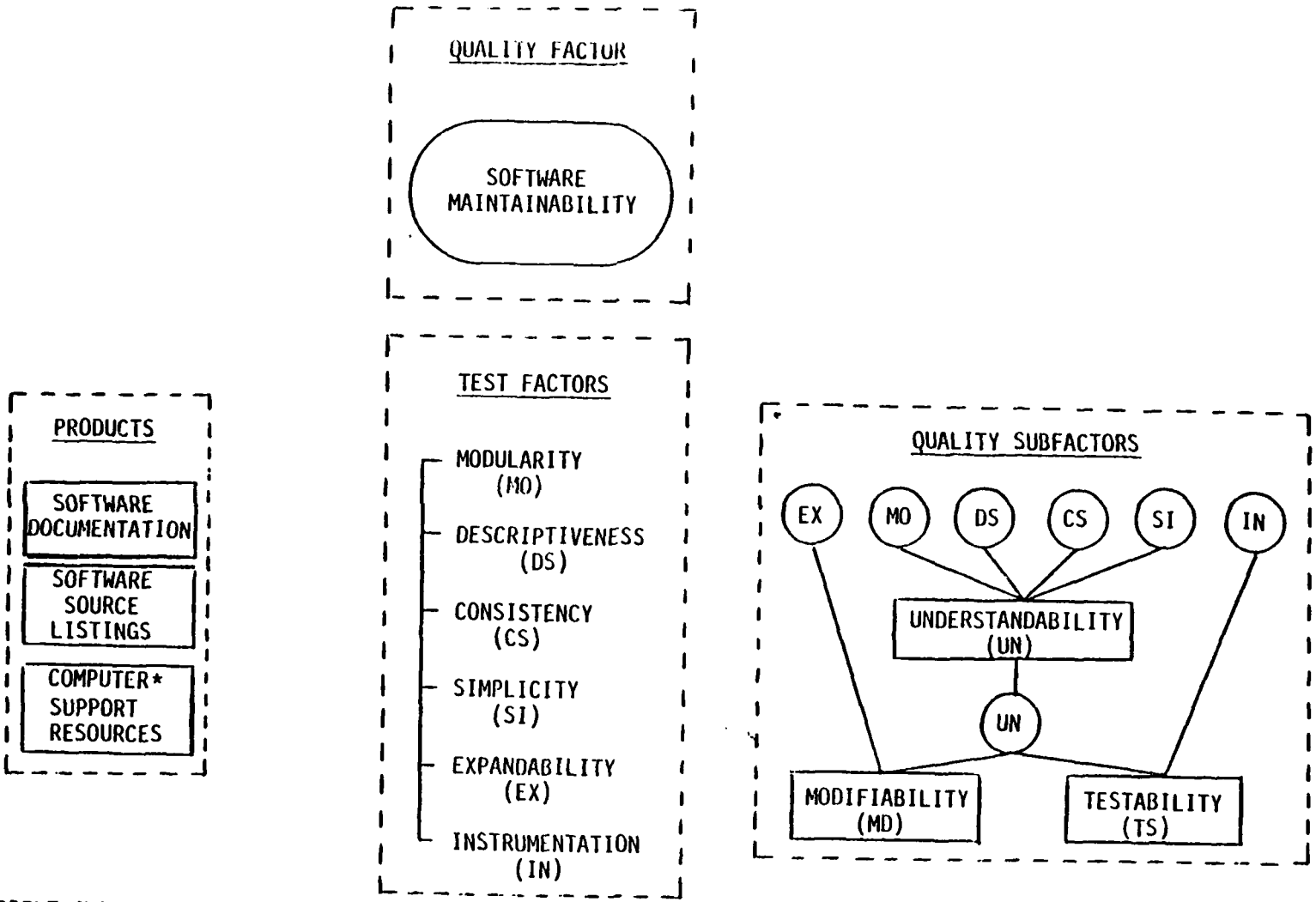


Figure I-2. Computer System Hierarchy



*NO CURRENT EVALUATION QUESTIONNAIRE

Figure I-3. Element of Software Maintainability

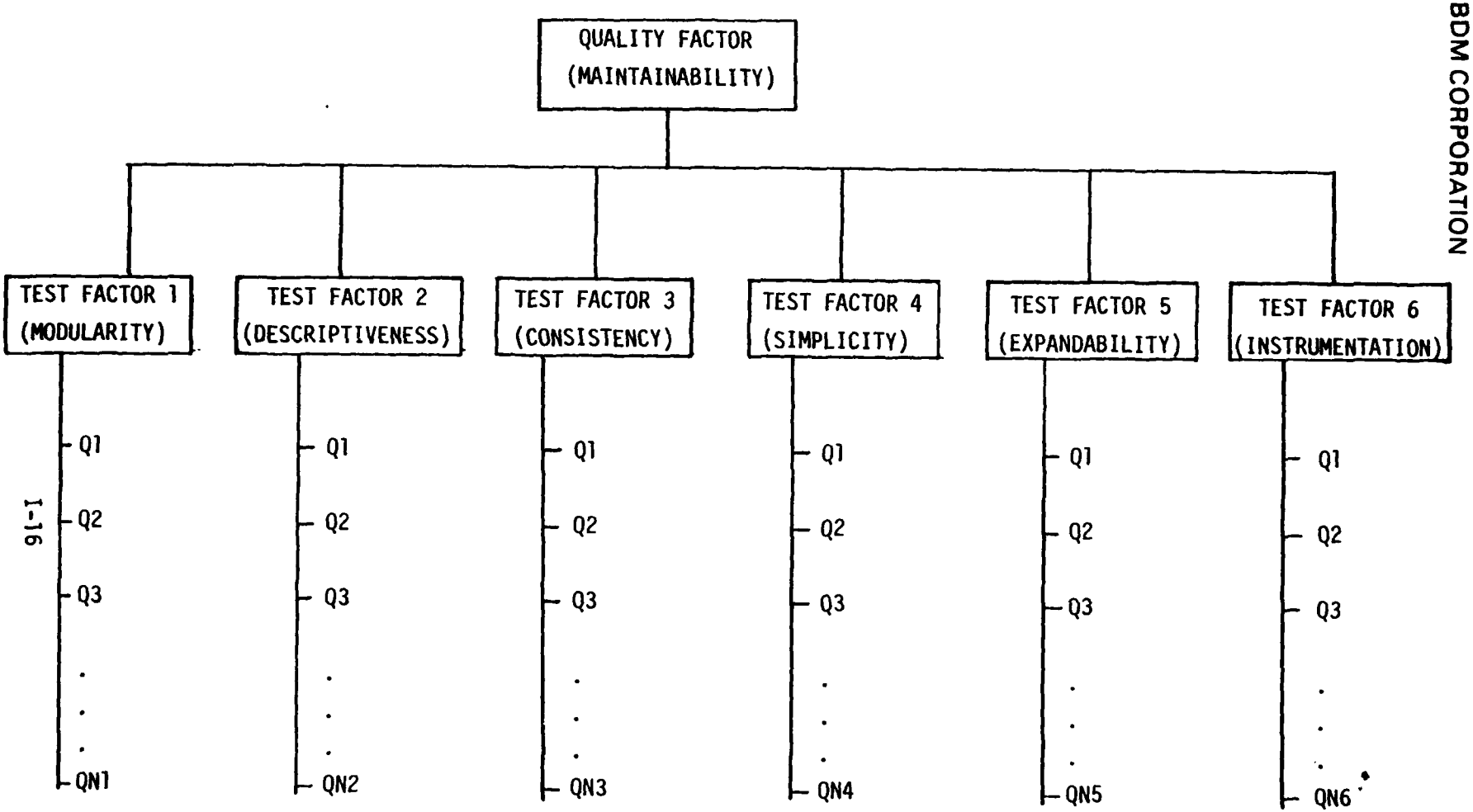


Figure I-4. Product Questionnaire Format

TABLE I-1. List of Typical Documentation Products

	LONG LIFE/HIGH COST SOFTWARE SYSTEMS	SHORT LIFE/LOW COST SOFTWARE SYSTEMS
Specifications	System Requirements Specification Standards & Conventions Documentation Plan Data Base Management Plan Preliminary Design Specification Interface Control Document Preliminary Design Review Material Detailed Design Spec (Build to) Critical Design Review Material Detailed Design Spec (Built to) Validation & Acceptance Test Plan Operator Interface Document	Preliminary Design Spec Detailed Design Spec (Build to) Detailed Design Spec (Built to) User's Guide
Configuration Control Forms	Design Problem Report Software Problem Report Documentation Update Transmittal Compool Change Request Data Base Change Request Modification Transmittal Memo	Revised Version Document

TABLE I-2. Definitions

SOFTWARE: Software consists of the programs and documentation which result from a software development process.

SOFTWARE MAINTAINABILITY: Software maintainability is a quality of software which reflects the effort required to perform the following actions:

(1) removal/correction of latent errors
(2) addition of new features/capabilities
(3) deletion of unused/undesirable features
(4) modification of software to be compatible with hardware changes.

Implicit in the above definition are the concepts that the software should be understandable, modifiable and testable in order to have effective maintainability.

UNDERSTANDABILITY: Software possesses the characteristics of understandability to the extent its purpose and organization are clear to the inspector.

MODIFIABILITY: Software possesses the characteristics of modifiability to the extent that it facilitates the incorporation of changes once the nature of the desired change has been identified.

TESTABILITY: Software possesses the characteristics of testability to the extent that it facilitates the establishment of verification criteria and supports evaluation of its performance.

TEST FACTORS: Software maintainability test factors are user-oriented general attributes of software which affect maintainability. The set of test factors includes: modularity, descriptiveness, consistency, simplicity, expandability, and instrumentation.

MODULARITY: Software possesses the characteristics of modularity to the extent a logical partitioning of software into parts, components, modules has occurred.

DESCRIPTIVENESS: Software possesses the characteristics of descriptiveness to the extent that it contains information regarding its objectives, assumptions, inputs, processing, outputs, components, revision status, etc.

CONSISTENCY: Software possesses the characteristics of consistency to the extent the software products correlate and contain uniform notation, terminology and symbology.

TABLE I-2. Definitions (Concluded)

SIMPLICITY: Software possesses the characteristics of simplicity to the extent that it lacks complexity in organization, language, and implementation techniques and reflects the use of singularity concepts and fundamental structures.

EXPANDABILITY: Software possesses the characteristics of expandability to the extent that a physical change to information, computational functions, data storage or execution time can be easily accomplished.

INSTRUMENTATION: Software possesses the characteristics of instrumentation to the extent it contains aids which enhance testing.

SOFTWARE DOCUMENTATION: Software documentation is the set of requirements, design specifications, guidelines, operational procedures, test information, problem reports, etc. which in total form the written description of the program(s) output from a software development process.

SOFTWARE SOURCE LISTINGS: Software source listings are the implemented representation (listing) described through a source computer language of the program(s) output from a software development process.

COMPUTER SUPPORT RESOURCES: Computer support resources include all the resources (software, computer equipment, facilities, etc.) which support the software being evaluated. (The documentation on the use of such resources to support program maintenance is the product to be evaluated.)

PROGRAM: A program is a set of hierarchically related modules which can be separately compiled, linked, loaded and executed.

MODULE: A module is a set of "contiguous" computer language statements which has a name by which it can be separately invoked.

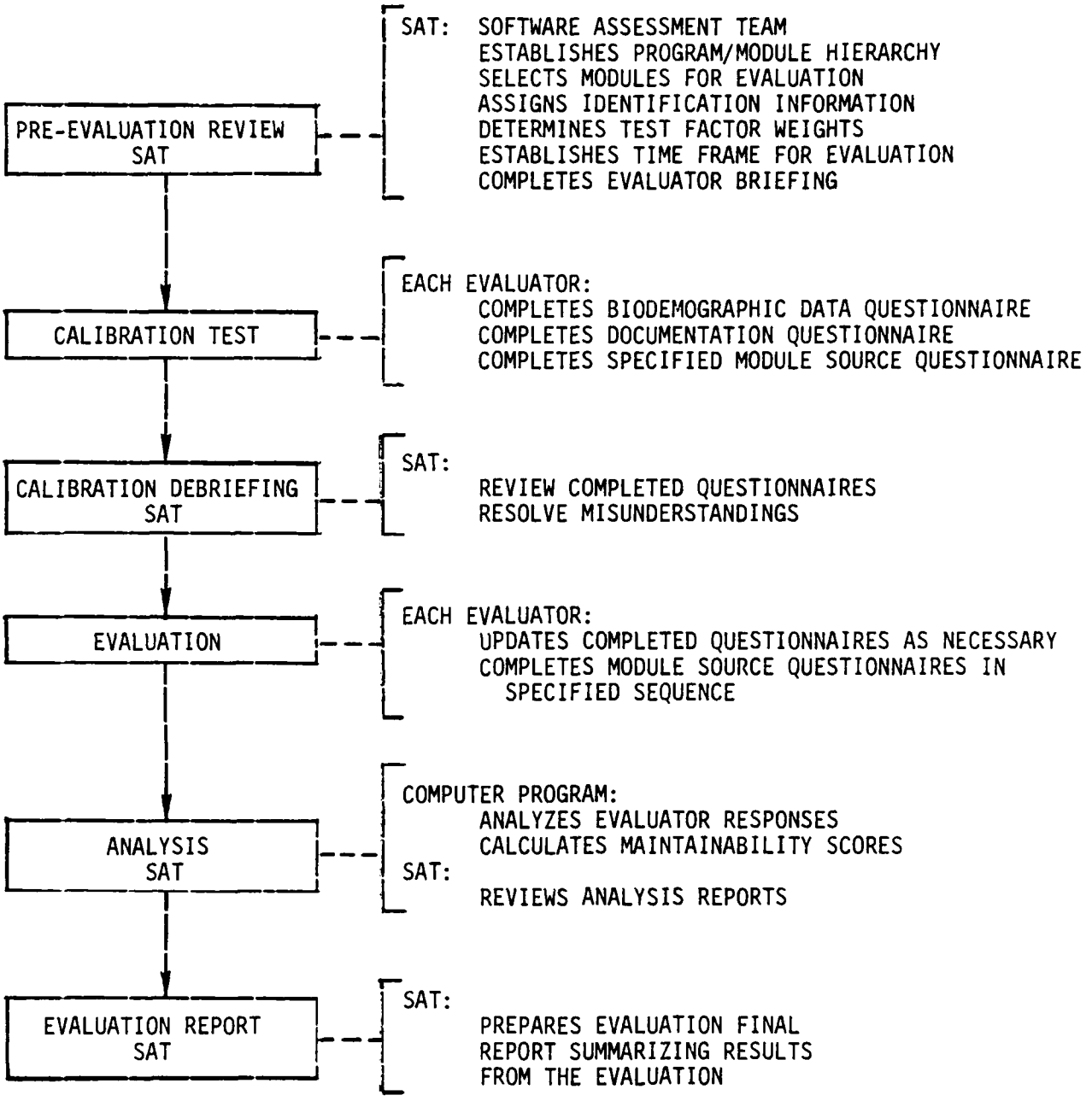


Figure I-5. Maintainability Evaluation Procedure

SECTION II
QUESTIONNAIRE RESPONSE GUIDELINES

A. GENERAL GUIDELINES

This section includes specific information for the evaluator concerning the evaluation of a software program for maintainability.

There are three software products which are evaluated as part of this evaluation: program documentation, module source listings, and documentation on the use of computer support resources. The evaluator completes one Software Documentation Questionnaire for program documentation and one Module Source Listing Questionnaire for each program module being evaluated. At present there is no questionnaire for the computer support resources and this aspect of the overall software maintainability evaluation will not be further considered in this handbook. Copies of the *Software Documentation Questionnaire* and the Module Source Listing Questionnaire are contained in Section III of this handbook.

There are several phases to the maintainability evaluation procedure as illustrated in figure I-4 (Section I). During the Pre-evaluation Review, the Software Assessment Team (SAT) evaluation coordinators brief the evaluators on the use of the questionnaires and the appropriate response forms. The questions are reviewed in order to minimize evaluator misunderstandings and to provide any particular "tailoring" of the evaluation to the particular program being evaluated. At the end of this briefing the evaluators complete the Software Documentation Questionnaire and one Module Source Listing Questionnaire (on the selected module). In addition, the evaluator completes a questionnaire concerning certain background experience (biodemographic questionnaire). At the end of this Calibration phase, the SAT evaluation coordinators and the evaluators get together for a debriefing. This debriefing centers around problems, misunderstanding, disagreement, etc. that the evaluators

had completing the two questionnaires. After this Calibration Debriefing, the evaluator is free to rework the completed questionnaires as necessary and complete the Module Source Listing Questionnaire on the rest of the modules being evaluated. Table II-1 summarizes some general evaluator guidelines for the Calibration and Evaluation phases.

After all questionnaires have been completed by the evaluators and collected, the SAT evaluation coordinators analyze the results to determine particular program maintainability problems as indicated by the numerical responses and the evaluator comments. Special attention is directed to evaluator disagreement and question reliability. The results of the evaluation are included as part of the overall system evaluation report.

The specific guidelines to filling out the response forms along with examples from a sample program evaluation are contained in the following subsections. Section II.E contains examples of program-specific information which will be included as a separate handout to the evaluator at the time of the Calibration Test.

B. EVALUATOR BIODEMOGRAPHIC RESPONSE GUIDELINES

The Biodemographic Questionnaire is completed during the Calibration phase of the evaluation procedure. This questionnaire helps the SAT in the analysis of the program evaluation. It in no way is intended to reflect either favorably or to the detriment of any evaluator, but is simply information on an evaluator's background education and experience. Over time, this information may help to determine when certain backgrounds correlate with certain response profiles.

An example of a completed biodemographic questionnaire is given in Section II.D. Since the data from this form must be manually keypunched, it is preferable that the "style" shown in the example form be copied as closely as possible.

General 1. Work independently. 2. Complete questionnaires in specified sequence. 3. Answer all questions.
Calibration Test 1. Complete questionnaires in the sequence: a. Software Documentation Questionnaire b. Module Source Listing Questionnaire 2. The specific module to be evaluated is noted in the evaluation handout along with the list of all modules selected for evaluation. All necessary identification information is also contained in the handout.
Evaluation 1. Rework responses as necessary on questionnaires completed as part of the Calibration Test. 2. Complete questionnaires on the remaining modules in the sequence specified in the evaluation handout. 3. Take care to correctly complete all information on the response forms. 4. Carefully observe the specific response guidelines contained in Section IV of this handbook.

TABLE II-1. General Evaluator Guidelines

C. PROGRAM EVALUATION RESPONSE GUIDELINES

The evaluators should use the Software Documentation Questionnaire and the Module Source Listing Questionnaire only to obtain the specific questions. The evaluator answers to the questions should be entered on an AFTEC Form 92, Questionnaire Answer Sheet. One Form 92 should be completed for each use of a Questionnaire. If comments are appropriate for a given question, then AF Form 1530, Punch Card Transcript, should be completed.

The specific details of using the Questionnaires and Forms are explained in the next subsections. Examples are contained in Sections II.D and II.E.

1. Response Forms

There are two forms on which an evaluator can record his responses to questions: AFTEC Form 92 for answers, and AF Form 1530 for associated comments. Form 92 is processed through an AFTEC optical scanner so it is important that all evaluator input in the oval fields be dark and that no extraneous marks appear. Errors should be completely erased. Form 1530 is processed manually through keypunch operations. Since the data will be keypunched as entered on the form, the evaluator should exercise care in entering data in the correct fields. It should also be explained that the Form 92 contains no explanatory information since it was designed to be a general form for use by any group using questionnaire forms. There are three blocks on Form 92: Descriptive Identification Block, Numerical Identification Block, and Evaluator Response Block.

The Descriptive Identification Block contains information which identifies the particular questionnaire type, system, subsystem, program, module, evaluator, date and time to complete. This block is only used for a visual identification check, if necessary, and is not processed by the optical scanner. The suggested format with an example is given in Section II.D.

The Numerical Identification Block contains numeric codes for the same information contained in the Descriptive Identification Block. The numeric codes are entered in the appropriate column fields and the associated numbered ovals are darkened. Extreme care should be taken to enter all data in this block correctly since this block is optically scanned and is effectively the only output information which uniquely matches the evaluator responses with the correct evaluator and software program information. The required format with an example is given in Section II.D. Since the only fields which might change from questionnaire to questionnaire for a given evaluator are fields 1 (Type of Questionnaire), 5 (Module Number), 7 (Day), 8 (Month), and 10 (Time), it is suggested that a template be used to complete the numerical entries. Such a template is illustrated in Section II.D.

2. Response Scale

The following response scale should be used to answer each question:

A. COMPLETELY AGREE
B. STRONGLY AGREE
C. GENERALLY AGREE
D. GENERALLY DISAGREE
E. STRONGLY DISAGREE
F. COMPLETELY DISAGREE

In addition, one or more of the following standardized comment responses can be selected:

I. I had difficulty answering this question.
J. A written comment is recorded on AF Form 1530, Punch Card Transcript.

The responses G and H do not currently have any meaning. The responses A to F indicate the extent to which the evaluator agrees/disagrees with the question statement. Specific response instructions are included with the question and/or with the question discussion given in Section IV.

Occasionally the evaluator may find it difficult to answer a question because it "doesn't seem to apply" to the program being evaluated. Most of the time this situation arises when the particular software requirements do not involve the need for a certain undesirable (relative to maintainability) software characteristic; or, the intrinsic nature of the software (e.g., language used) eliminates the possibility that such an undesirable software characteristic can exist. As an example, a particular module may not have any statement labels. A question statement such as "Statement labels have been named in a manner which facilitates locating a label in the source listing." addresses a software characteristic (descriptive labels) which because the software has no labels is somewhat obviated. However, note that the software will be more understandable since no branching to labeled statements can occur. Since understandability is a higher goal the evaluator should give as high a response (A) as if there were labels all very well named.

Sometimes the situation arises where the question implies the existence of a particular item (chart, matrix, etc.) and the question asks about the content within the item. If the item does not exist, then this indicates a serious defect (relative to maintainability) of the software. In this case, the evaluator response should be F. Questions where this situation may arise more frequently will contain instructions to so mark the response (or perhaps temper the response). Questions where this situation might arise but less frequently will have special response instructions included in Section II.D.

It is emphasized that responses of A or F are in general not expected. These responses indicate a best possible or worst possible characteristic relative to software in general. There will be obvious cases where an A or an F answer is the correct answer. In most non-obvious cases an answer of A or F should be relatively infrequent.

3. Software Documentation Questionnaire

This questionnaire is used to evaluate the overall format and content of the documentation (not including source listings) for the program being evaluated. The Table I-2 (Section I) lists typical docu-

mentation which might be related to a program. Although the information required to answer the Software Documentation Questionnaire questions may be spread out among several distinct documents, the primary information sources which are always considered a part of the evaluation are the program functional/detailed design specifications and the program maintenance/operational procedures.

Section II.E gives an example of how the documentation which is to be generally included as source for possible documentation question answers is specified to the evaluator. The scores on this questionnaire form the raw score for the Software Documentation part of the maintainability evaluation. Section II.E as applied to the specific program being evaluated will be a separate handout to the evaluator at the time of the Calibration Test.

4. Module Source Listing Questionnaire

This questionnaire is used to evaluate the overall format and content of the source listing for the program module being evaluated, and to evaluate the consistency between the module's documentation and source listing.

Section II.E gives an example of how the program modules which are to be evaluated are specified to the evaluator. One questionnaire per module is completed by the evaluator. The cumulative scores over all modules evaluated form the raw score for the Software Source Listing part of the maintainability evaluation. Section II.E as applied to the specific program being evaluated will be a separate handout to the evaluator at the time of the Calibration Test.

All evaluator responses to module source listing questions are recorded on one AFTEC Form 92, Questionnaire Answer Sheet (one Form 92 per module).

D. EXAMPLE RESPONSE FORMS

The figures on the following pages contain response form instructions and examples as summarized below:

The following format is suggested for completing the Descriptive identification block on the questionnaire response forms.

DESCRIPTIVE IDENTIFICATION
<QTYPE>
<SYSTEM>
<SUBSYSTEM>
<PROGRAM>
<MODULE>
<EVALUATOR>
<DATE>
<TIME>

<QTYPE>	=	Questionnaire Type	= Q1 (documentation) or Q2 (module)
<SYSTEM>	=	System Name	= B52CPT (example)
<SUBSYSTEM>	=	Subsystem Name	= G (example)
<PROGRAM>	=	Program Name	= RTOP (example)
<MODULE>	=	Module Name	= [See module name list II.E]/DOCMNT
<EVALUATOR>	=	Evaluator's Last Name	= [See evaluator name list II.E]
<DATE>	=	Date Questionnaire Form Started	= <day><month><year>
<TIME>	=	Time to Complete Questionnaire Form Rounded to Nearest Hour	= <time in integer hours>

NOTE: DOCMNT is entered in the <MODULE> field if the questionnaire type is Q1.

Figure II-1. Descriptive Identification Block Format

DESCRIPTIVE IDENTIFICATION
Q2
B52CPT
G
RTOP
MODX44
MERRY
02 MAY 78
03

Figure II-2. Descriptive Identification Block Example

The following format is required for completing the Numerical Identification block on the questionnaire response forms. The correct numerical integers should be input in the appropriate fields. Extreme care should be taken in entering this data and in completely covering the associated numbered oval in each column. This numerical ID is processed by an optical scanner and is effectively the only way that the questionnaire responses can be correlated with the system/subsystem/program.../evaluator/....

FIELD	1	2	3	4	5	6	7	8	9	10
COLUMN	1	2,3	4,5	6,7	8,9	10,11,12	13,14	15,16	17,18	19,20

FIELD	COLUMNS	DATA DESCRIPTION	GENERAL RANGE
1	1	Type of Questionnaire	0-9
2	2,3	System	00-99
3	4,5	Subsystem	00-99
4	6,7	Program	00-99
5	8,9	Module Number	00-00
6	10,11,12	Evaluator Number	000-999
7	13,14	Day Date	01-31
8	15,16	Month Response Form	01-12
9	17,18	Year Started	78-99
10	19,20	Time (hrs) to Complete Questionnaire	00-99

NOTES:
1) Field 1: Enter 1 for documentation, 2 for module.
2) Field 5: 00 is entered if questionnaire form is documentation; only module numbers allowed are shown in II.E.

Figure II-3. Numerical Identification Block Format

DATA	FIELD	DESCRIPTIVE	NUMERICAL
	Questionnaire Type	Q2	2
	System	B52CPT	01
	Subsystem	G	01
	Program	RTOP	01
	Module Name/Number	MODX44	44
	Evaluator Name/Number	MERRY	004
	Date: Day	02 May 78	02
	Month		05
	Year		78
	Time (hrs)	03	03

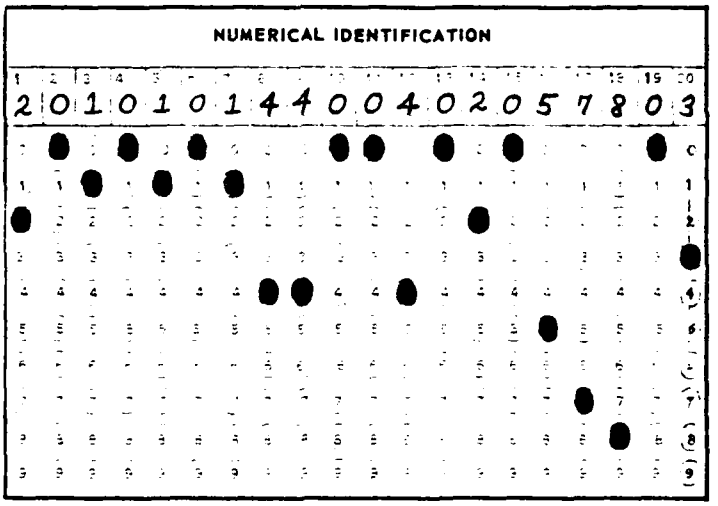
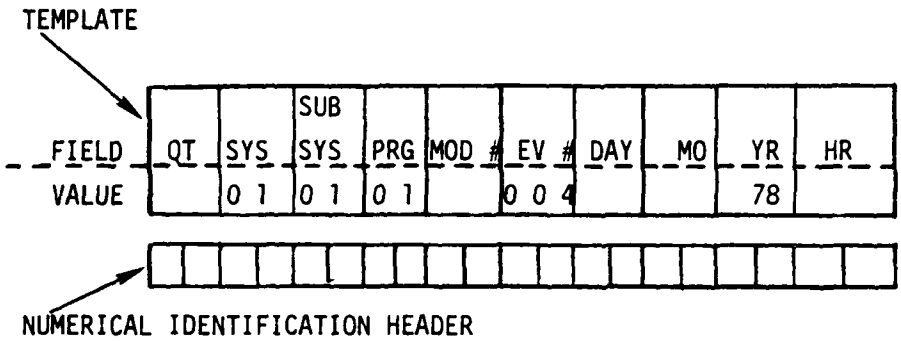


Figure II-4. Numerical Identification Block Example



NOTE: Fields QT, MOD#, DAY, MO, HR may vary for the given evaluator (004).

Figure II-5. Numerical Identification Block Template Example

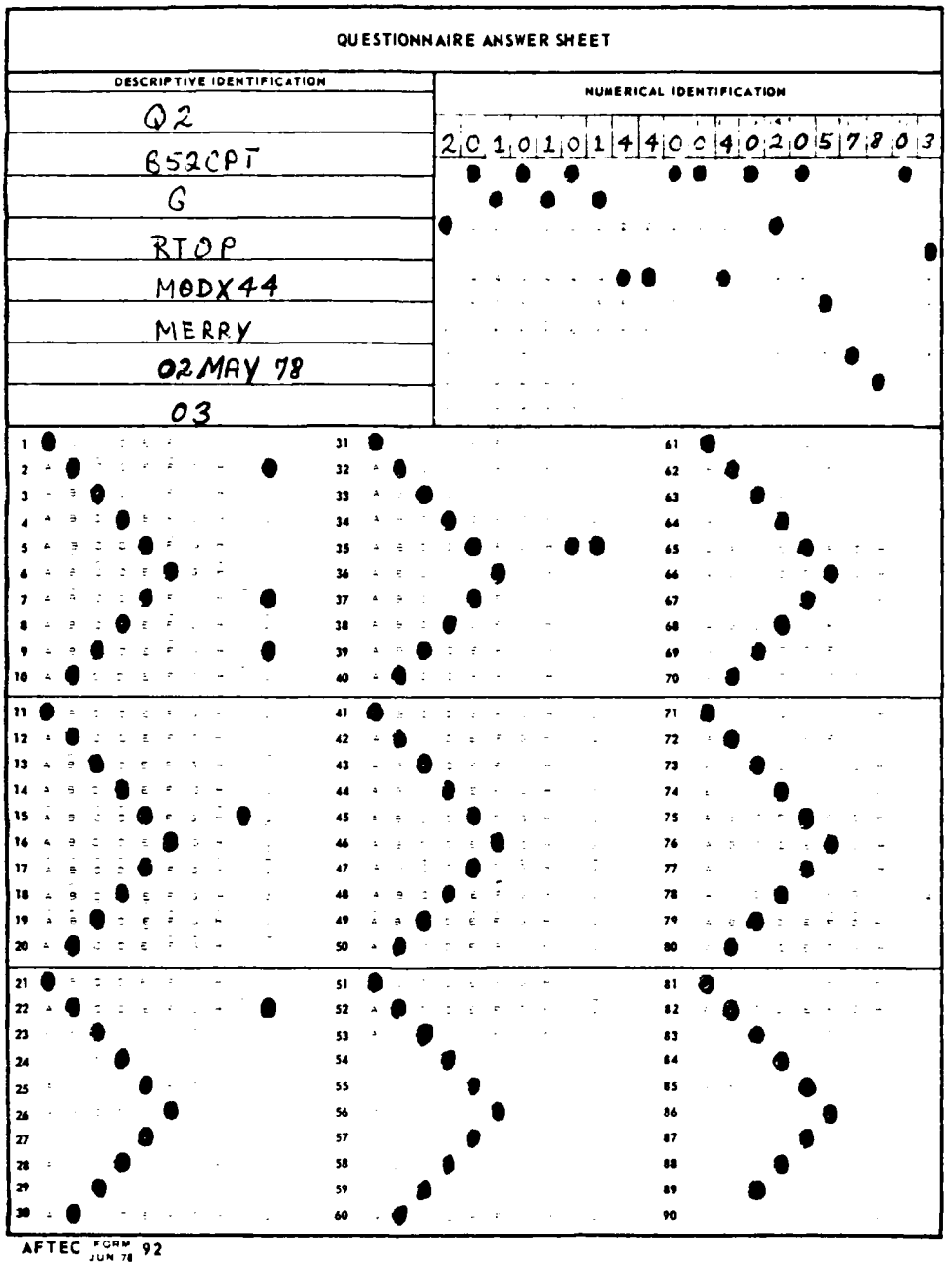


Figure II-6. AFTEC Form 92 Example

ROW 1: Enter Numerical Id in columns 1-20.
Enter module name or DOC beginning column 30

ROW 21: Enter question number in column 4-5 (right justify)
Enter Comment in Columns 11-80

Further guidelines are described in the illustrative comments below.

```
2010101440040205780 3                MODX44

    02      THIS IS A COMMENT ON QUESTION NUMBER 2. THIS IS ALSO AN ILLUSTRATION
            OF THE CONTINUATION OF A COMMENT ACROSS ONE OR MORE ROWS.

    09      EVALUATORS ARE ENCOURAGED TO MAKE COMMENTS. FOR VISUAL AID, BLANK
            LINES MAY BE USED TO SEPARATE QUESTION COMMENTS. THE ONLY REAL
            RESTRICTIONS ARE THAT THE COMMENTS PHYSICALLY RESIDE BETNEEN COLUMN 11
            AND COLUMN 80, AND THAT THE FIRST LINE(AT LEAST) HAVE THE QUESTION
            NUMBER IN COLUMNS 4 AND 5(RIGHT JUSTIFIED PLEASE).

    07      THIS COMMENT ILLUSTRATES THAT SEQUENCING COMMENTS IS NOT NECESSARY.

    22      THE EVALUATOR IS ENCOURAGED TO COMMENT WHEN AN EXPLANATION OF THE
            ASSOCIATED RESPONSE NOULD BE USEFUL, OR WHEN IT IS DESIRABLE TO
            COMMENT ON THE GENERAL WORTHLESSNESS OR EXCELLENCE OF THE QUESTION
            ITSELF.

    35      FOR IDENTIFICATION PURPOSES, THE ROW 1(NUMERICAL ID) SHOULD BE
            REPEATED IF MORE THAN ONE AF FORM 1530 IS NEEDED FOR COMMENTS
            RELATED TO ONE QUESTIONNAIRE.
```

Figure II-7. Instructions and Format for AF Form 1530

EVALUATOR BIODEMOGRAPHIC QUESTIONNAIRE

1. Will you be maintaining any portion of the software in this system?

   YES ☒ NO ☐

2. Please provide the information requested in the following table. Enter in the "skill level" block one of the following numbers to indicate the degree of your familiarity with the area. Enter the number that reflects your highest level of skill. If you have no familiarity with an item, please leave it blank.

   1 = Have studied - classroom or on-the-job training.

   2 = Working knowledge - limited on-the-job or project experience in computer work.

   3 = Used extensively - can independently perform full range of job requirements.

TYPE OF EXPERIENCE	MONTHS OF EXPERIENCE	MOST RECENT EXPERIENCE (MO./YR.)	SKILL LEVEL	COMMENTS
DEVELOPMENT PROGRAMMING				
HOL	60	06/78	3	BASIC(3), FORTRAN(3), PASCAL(2)
ASSEMBLY	120	06/78	3	CDC-924/A, IBM/360, HARRIS/5
REAL TIME	84	06/78	3	CDC-924/A
NON REAL TIME	120	06/78	3	CDC-924/A, IBM/360, HARRIS/5
SOFTWARE SYSTEMS ANALYST				
HOL	60	06/78	3	BASIC(2), FORTRAN(1), PASCAL(1)
ASSEMBLY	60	06/78	3	CDC-924/A, IBM/360, HARRIS/5
REAL TIME	48	06/78	3	CDC-924/A
NON REAL TIME	84	06/78	3	CDC-924/A, IBM/360 HARRIS/5

Figure II-8. Evaluator Biodemographic Questionnaire Example

TYPE OF EXPERIENCE	MONTHS OF EXPERIENCE	MOST RECENT EXPERIENCE (MO./YR.)	SKILL LEVEL	COMMENTS
SOFTWARE MAINTENANCE				
HOL	1	04/78	1	FORTRAN
ASSEMBLY	84	06/78	3	CDC-924/A, IBM/360, HARRIS/6
REAL TIME	84	06/78	3	CDC-924/A
NON REAL TIME	84	05/78	3	CDC-924/A, IBM/360, HARRIS/6
SOFTWARE MANAGER	12	05/78	2	
SYSTEM OPERATOR	120	06/78	3	
COMPUTERS (indicate types and your capacity in the comments column, e.g. 1401 programmer)	54	09/72	3	Software Develop./Maint. } CDC-924/A
			3	Hardware Maintenance
			3	Operation
	48	06/78	3	Software Development } HARRIS/6
			3	Operation
	12	04/75	2	Software Development } IBM/360
COMPUTER LANGUAGES	60	06/78	3	BASIC
	12	06/78	2	FORTRAN
	12	06/78	2	PASCAL
	120	06/78	3	ASSEMBLY

Figure II-8. Evaluator Biodemographic Questionnaire Example (Continued)

3. Have you received any formal training related to the software being evaluated?

YES ☐ NO ☒

If you have received formal training, identify formal training by course and indicate the time spent in training.

COURSE TITLE	COURSE DURATION

4. Have you had any specific experience directly related to the software being evaluated?

YES ☒ NO ☐

If you have specific experience please indicate the type and lengths of experience.

TYPE OF EXPERIENCE	LENGTH OF EXPERIENCE
Software Development	10 months
Software Maintenance	18 months

Figure II-8. Evaluator Biodemographic Questionnaire Example (Continued)

5. Do you feel that your experience/background is sufficient to adequately complete the following?

   a. Program Design Questionnaire.

      YES ☒ NO ☐

   b. Module Design Questionnaire.

      YES ☒ NO ☐

6. Check all boxes below indicating the kinds of documents which you have written or participated in writing.

   ☐ Research Publication

   ☒ Users Guide/Manual

   ☒ Proposal

   ☒ Design Specification

7. Enter your years of education by type.

   Trade School ___ Years

   Military Technical 2 Years

   College 8 Years

8. Give your major field of study in school.

   Computer Science

9. List your GS level (Civilian) GS-13

   or Rank (Military) _____

Figure II-8. Evaluator Biodemographic Questionnaire Example (Continued)

IMPORTANCE OF SOFTWARE MAINTAINABILITY FACTORS

Factors listed below will influence the maintainability of software. Rate the importance of each factor to program maintenance and modification. Use a scale from 1 (very little importance) to 10 (very great importance) on each item. Each factor is to be rated independent of the other factors. Thus several factors could be assigned the same importance number.

DOCUMENTATION

Modularity	8
Descriptiveness	10
Consistency	6
Simplicity	7
Expandability	5
Instrumentation	6

SOURCE LISTING

Modularity	10
Descriptiveness	9
Consistency	7
Simplicity	8
Expandability	5
Instrumentation	6

Figure II-8. Evaluator Biodemographic Questionnaire Example (Concluded)

E. EXAMPLE PROGRAM DEPENDENT DATA

This section contains examples of program dependent data that will be separately given to evaluators at the time of the Calibration Test. There are four items of program dependent data as summarized below:

(1) Software Maintainability Evaluation Schedule

(2) Software Documentation Evaluation List

(3) Software Module Name/Number List

(4) Evaluator Name/Number List

On the following pages are figures of these four items.

The following schedule is planned for the software maintainability of the {program name to be supplied by SAT}.

Evaluation Milestone	Date
Begin Calibration Test	
Conduct Calibration Debriefing	{To be supplied by SAT}
Finish Software Maintainability Evaluation	

Notes:

{Notes on personnel involved in Evaluation Milestones may be included here}

Figure II-9. Software Maintainability Evaluation Schedule

The following documentation will form the basis of the software maintainability evaluation of the B52CPT Real Time Operating Program (RTOP), version G.

Document	Description
Vol. I	Programmer's Reference Manual
Vol. II	Operator's Reference Manual
Vol. III	Executive Program
Vol. IV	Graphics
Vol. V	Module Desc: Aerodynamics, Engines, Flight Controls, Environment
Vol. VI	Module Desc: Aircraft Systems
Vol. VII	Glossary and Data Base

Figure II-10. Software Documentation Evaluation List

The following modules have been randomly selected to be evaluated for the software maintainability evaluation of the B52CPT Real Time Operating Program (RTOP), version G. The program RTOP consists of thirty-six modules plus the executive. The modules have names MODG01, MODG02, ..., MODG48 (some names are spares) and SMB52G for the executive. Corresponding names MODX01, MODX02, ..., MODX48 and MODX49 for the executive have been chosen for use on this evaluation. The modules should be evaluated in the sequence as listed below.

MODULE NAME	MODULE NUMBER	MODULE DESCRIPTION
MODX05	05	Crash
MODX09	09	NAV Instruments
MODX27	27	Call Letter Keyer
MODX08	08	Environmental Effects: Aural
*MODX44	44	Communications (UHF, Liaison Radio)
MODX26	26	Flap and Drag Chute
MODX28	28	TACAN
MODX41	41	Mass, Moments of Inertia, CG Position
MODX49	49	SMB52G Executive

*This module has been selected to be evaluated during the Calibration Test.

Figure II-11. Module Name/Number List

The following evaluator names and numbers are to be used to complete information in the descriptions and numerical identification blocks.

EVALUATOR LAST NAME	EVALUATOR NUMBER
BERGSTROM	001
CLARK	002
GIMENEZ	003
MERRY	004
WILSON	005

Figure II-12. Evaluator Name/Number List

SECTION III
QUESTIONNAIRES

A. SOFTWARE DOCUMENTATION QUESTIONNAIRE

The following pages of this section contain the Software Documentation Questionnaire.

SOFTWARE DOCUMENTATION QUESTIONNAIRE
INSTRUCTIONS

The only material required in order to complete this questionnaire is the program documentation (no source listings). The evaluator should consult the evaluator guideline handbook for more detailed instructions on completing this questionnaire and interpreting the questions.

This questionnaire contains questions which address some of the characteristics of a software program's documentation which affect the maintainability of the program. There are six maintainability factors: modularity, descriptiveness, consistency, simplicity, expandability, and instrumentation. Questions concerning each of these factors have been grouped together.

The evaluator should first complete all the required questionnaire identification information on an AFTEC Form 92, Questionnaire Answer Sheet. The evaluator should then read each question, refer to the program's documentation as necessary, and record on the answer sheet the one response which indicates the most appropriate level of agreement or disagreement with the question statement. The following response scale is to be used for all questions:

A. COMPLETELY AGREE
B. STRONGLY AGREE
C. GENERALLY AGREE
D. GENERALLY DISAGREE
E. STRONGLY DISAGREE
F. COMPLETELY DISAGREE

In addition, one or more of the following standardized comment responses can be selected:

I. I had difficulty answering this question.
J. A written comment is recorded on AF Form 1530, Punch Card Transcript.

MODULARITY QUESTIONS

Format Modularity

Note: The following questions relate to how the documentation has been physically formatted into functional parts.

1. Program documentation includes a separate part for the description of program external interfaces.
2. Program documentation includes a separate part for the description of each major program function.
3. Program documentation includes a separate part for the description of the program global data base.
4. Major parts of the program documentation are essentially self-contained.
5. Program documentation has been physically separated into (sets of) volumes each with a distinct function.

Data Modularity

Note: The following questions relate to how the program data base has been designed for functional use.

6. Each global data structure is partitioned into functionally related sets of variables.
7. Data storage locations are not used for more than one type of data structure.

Processing Modularity

Note: The following questions relate to how the program control and data flow has been designed for functional use.

8. The program control flow is organized in a top down hierarchical tree pattern.
9. Program initialization processing is done by one (set of) modules(s) designed exclusively for that purpose.
10. Program termination processing is done by one (set of) module(s) designed exclusively for that purpose.
11. Program I/O is done by one (set of) module(s) designed exclusively for that purpose.
12. Program error processing is done by one set of modules designed exclusively for that purpose.

DESCRIPTIVENESS QUESTIONS

Format Descriptiveness

Note: The following questions relate to how the format of the documentation assists in locating or clarifying program information.

13. Each physically separate part of the documentation includes a useful table of contents.
14. Each physically separate part of the documentation includes a glossary of major terms and acronyms.
15. Each physically separate part of the documentation includes an index.
16. It is easy to locate specific information within the documentation.
17. The documentation includes a current version description document.
18. A master list which identifies all software documentation is available.

Constraints Descriptiveness

Note: The following questions relate to program constraints/requirements.

19. Any dynamic allocation of resources (storage, timing, priority, hardware services, etc.) is explained in the documentation.
20. Timing requirements for each major function of the program are explained in the documentation. (Answer A if program is non-real time).
21. Storage requirements for each major function of the program are explained in the documentation.

Module Descriptiveness

Note: The following questions relate to the general explanation of program components (modules).

22. The inputs to each module are explained in the documentation.
23. The processing of each module is explained in the documentation.
24. The outputs from each module are explained in the documentation.
25. Special processing considerations (error, interrupt, etc.) of each module are explained in the documentation.
26. There is a flow chart (or equivalent) for each module which diagramatically illustrates the inputs, general processing, and outputs for the module.

DESCRIPTIVENESS QUESTIONS (Continued)

External Interface Descriptiveness

Note: The following questions relate to the description of external program interfaces.

27. Program initialization and termination processing is explained.

28. Recovery from externally generated error conditions which could affect the program is explained.

29. The process of recovering from internally generated error conditions is explained.

30. Input of program data is explained.

31. Output of program data is explained.

Internal Interface Descriptiveness

Note: The following questions relate to the general description of internal program interfaces.

32. There is a set of charts which show the general program control and data flow hierarchy among all modules.

33. There is a master list (chart, table, section, etc.) identifying where each global variable is used.

34. The master list (referenced above) includes information about each global variable such as type, range, scaling, units, etc.

Math Model Descriptiveness

Note: The following questions relate to the description of any significant mathematical techniques, algorithms or models.

35. The use of any complex mathematical model (technique, algorithm) is explained in the documentation.

36. The documentation on each complex mathematical model includes information such as a derivation, accuracy requirements, stability considerations and references.

CONSISTENCY QUESTIONS

Format Consistency

Note: The following questions relate to the physical consistency of the documentation.

37. It appears that a set of standards has been followed for the development of the program documentation.

38. It appears that a set of standards has been followed for the construction of the program and module flow charts (or equivalent).

39. Documentation of each major functional part of the program follows the same format.

40. The format of the documentation reflects the organization of the program.

Design Consistency

Note: The following questions relate to general design decisions which reflect consideration of consistency.

41. It appears that programming conventions have been established for the interfacing of modules (e.g., no global data arguments, no constant arguments, specific syntax in the case of assembly, input/output arguments clearly distinguished, etc.).

42. It appears that programming conventions have been established for I/O processing (e.g., which modules perform I/O, I/O type: random, sequential, formatted).

43. It appears that programming conventions have been established for error processing (e.g.,which modules perform error processing, error codes, format of error messages, method of recording and/or reporting errors, etc.).

44. A naming convention for modules appears to have been used.

45. A naming convention for global variables appears to have been used.

SIMPLICITY QUESTIONS

Format Simplicity

Note: The following questions relate to the simplicity of the documentation language and physical organization.

46. The terminology used in the documentation to describe the program is easily understood.
47. The documentation is physically organized as a systematic description of the program from levels of less detail to levels of more detail.
48. Each descriptive part (sentence, paragraph, subsection, section, chapter, volume, etc.) tends to express one central concern.
49. The amount of cross referencing among parts of the documentation contributes to the understandability of the program description.

Design Simplicity

Note: The following questions relate to the simplicity of the program design.

50. The program source language is a high order language (HOL).
51. The use of recursive/reentrant programming techniques is not excessive.
52. Each module of the program is designed to perform only one major function.
53. Resource (storage, timing, tape drives, disks, consoles, etc.) allocation is fixed throughout program execution.
54. The control flow among modules is easy to follow.
55. The timing scheme designed for the program is easily understood.
56. The program is designed so that modules are not interrupted during execution.
57. A knowledge of mathematics beyond basic algebra is not required to understand the mathematical functions performed by the program.

EXPANDABILITY QUESTIONS

Format Expandability

Note: The following questions relate to how the documentation has been physically formatted to allow for addition, change, or deletion of information.

58. A numbering scheme has been adopted which allows for easy addition or deletion of narrative parts of the documentation.

59. Graphic materials (figures, charts, lists, etc.) are physically separate (e.g., on separate pages) from narrative description.

60. A numbering scheme has been adopted which allows for easy addition or deletion of graphic materials (figures, charts, lists, etc.).

Design Expandability

Note: The following questions relate to how the program has been designed to allow for addition, change, or deletion of capabilities.

61. The program timing scheme appears to be flexible enough to allow for modification (e.g., reorganization, addition, deletion of functional parts).

62. There is a reasonable timing margin for each major program function (rate group, time slice, priority level, etc.).

63. Documentation narrative explains the procedure(s) for altering basic data storage sizes in order to accommodate increase or decrease.

64. The program has been designed to allow for an increase in storage utilized before the storage capacity is exceeded.

65. Those modules dependent upon data structure sizes are identified.

66. The program has been designed so that functional parts may be added or deleted.

INSTRUMENTATION QUESTIONS

Format Instrumentation

Note: The following questions relate to how the documentation has been physically formatted to aid in the program testing function.

67. There is a separate part of the documentation for the description of a program test plan.

68. There is a separate part of the documentation for the description of sample test data.

69. There is a separate part of the documentation for the description of program support tools which would aid in testing the program.

Design Instrumentation

Note: The following questions relate to how the program has been designed to include aids for testing the program.

70. A set of test procedures to be used for program check-out is explained.

71. The set of test procedures (referenced above) provides useful unit testing information.

72. The set of test procedures (referenced above) provides useful information on limitations/incompleteness.

73. The program has been designed with the capability to display test inputs and outputs in summary form.

74. A standardized set of program test data (input and output) has been designed to exercise the program.

75. The program has been designed to include test probes to aid in identifying processing performance.

76. Error checking within the program has been designed to include such features as diagnostic reporting, I/O parameter checking, runtime index range checking, etc.

GENERAL QUESTIONS

Note: The following questions relate to the evaluator's general impression of the program documentation contribution to program maintainability. The definitions of maintainability and the maintainability evaluation test factors as found in the evaluator guideline handbook should be reviewed before completing these questions.

77. Modularity as reflected in the program documentation contributes to the maintainability of the program.

78. Descriptiveness as reflected in the program documentation contributes to the maintainability of the program.

79. Consistency as reflected in the program documentation contributes to the maintainability of the program.

80. Simplicity of the program as reflected in the program documentation contributes to the maintainability of the program.

81. Expandability as reflected in the program documentation contributes to the maintainability of the program.

82. Instrumentation as reflected in the program documentation contributes to the maintainability of the program.

83. Overall it appears that the characteristics of the program documentation contribute to the maintainability of the program.

B. MODULE SOURCE LISTING QUESTIONNAIRE

The following pages of this section contain the Module Source Listing Questionnaire.

MODULE SOURCE LISTING QUESTIONNAIRE INSTRUCTIONS

The only material required in order to complete most of this questionnaire is the source listing for the module being evaluated. The documentation for the module will also be required for the consistency questions. The evaluator should consult the evaluator guideline handbook for more detailed instructions on completing this questionnaire and interpreting the questions.

This questionnaire contains questions which address some of the characteristics of a software module's source listing which affect the maintainability of the module. There are six maintainability factors: modularity, descriptiveness, consistency, simplicity, expandability, and instrumentation. Questions concerning each of these factors have been grouped together.

The evaluator should first complete all the required questionnaire identification information on an AFTEC Form 92, Questionnaire Answer Sheet. The evaluator should then read each question, refer to the module's source listing as necessary, and record on the answer sheet the one response which indicates the most appropriate level of agreement or disagreement with the question statement. The following response scale is to be used for all questions:

A. COMPLETELY AGREE
B. STRONGLY AGREE
C. GENERALLY AGREE
D. GENERALLY DISAGREE
E. STRONGLY DISAGREE
F. COMPLETELY DISAGREE

In addition, one or more of the following standardized comment responses can be selected:

I. I had difficulty answering this question.
J. A written comment is recorded on AF Form 1530, Punch Card Transcript.

MODULARITY QUESTIONS

Data/Control Modularity

Note: The following questions relate to the independence of each data and control structure.

1. Functionally related data elements have been organized into logical data structures.
2. The concepts of structured programming have been applied to the control structures in this module.
3. The use of data structuring techniques which involve the sharing (e.g., overlay, equivalence) of memory locations is not excessive.
4. The use of global data in this module is not excessive.

Processing Modularity

Note: The following questions relate to how the control flow of the module has been logically blocked.

5. The number of entry points of this module is not excessive.
6. The number of exit points of this module is not excessive.
7. This module performs only related functional tasks.
8. Each functional task of this module is an easily recognizable block of code.
9. It appears that each iteration block within this module has a single entry point.
10. It appears that each iteration block within this module has a single exit point.
11. It appears that each decision block within this module has a single entry point.
12. It appears that each decision block within this module has a single exit point.
13. When this module completes execution, control is returned to the calling module.
14. The use of the same variable for both input and output is not excessive in this module.

DESCRIPTIVENESS QUESTIONS

Preface Block Descriptiveness

Note: The following questions refer to a module preface block. If there is no preface block for this module, then mark a response of D, E, or F on the basis of whether the specified content of the preface block is clearly stated at any other place in the module.

15. Inputs to this module are described in a preface block.
16. Outputs from this module are described in a preface block.
17. The purpose of this module is described in a preface block.
18. Modules which call this module are identified in a preface block.
19. Modules which are called by this module are identified in a preface block.
20. Limitations (accuracy, timing, data I/O, etc.) are described as appropriate in a preface block.
21. Special processing (e.g., multiple entry/exit, error handling, algorithm peculiarities, etc.) is described in the preface block.
22. Documentation information (module name, programmer, algorithm references, revision data, etc.) is identified as appropriate in a preface block.

Imbedded Comments Descriptiveness

Note: The following questions relate to the effectiveness of the comments imbedded within a module's source listing.

23. The comments in this module contain useful information.
24. The quantity of comments does not detract from the legibility of the source listings (too few or too many comments may be considered to detract).
25. Transfers of control and destinations are clearly explained.
26. Machine-dependencies are clearly commented.
27. Comments describe each uniquely recognizable function of this module.
28. Attributes of each variable used in this module are described by comments and/or source language declarations.
29. Error processing/exits are clearly identified.

DESCRIPTIVENESS QUESTIONS (Continued)

Implementation Descriptiveness

Note: The following questions relate to the descriptiveness of the module organization.

30. It appears that a standard format for module organization has been followed within this module.

31. Variables are declared in a specification/declaration section.

32. Variable names are descriptive of their functional use.

33. The module code is indented within logical blocks to show control flow.

34. Statement labels have been named in a manner which facilitates locating a label in the source listing.

35. The machine cross reference listings appear to be useful.

CONSISTENCY QUESTIONS

External Consistency

Note: The following questions relate to the consistency between module documentation and the module source listing. Evaluators will need to consult both software products. "Flow chart" means flow chart or an equivalent form such as HIPO, PDL, etc. If there is no documentation narrative, flow chart or conventions as appropriate for the question, then mark a response of F.

36. This module's flow chart represents the logic control flow as shown in this module's source listing.

37. This module's flow chart represents the data flow as shown in this module's source listing.

38. The labels in this module's flow chart and the statement labels in this module's source listing are in agreement.

39. The inputs to this module as described in the documentation correspond to the inputs as shown in this module's source listing.

40. The outputs from this module as described in the documentation correspond to the outputs as shown in the module's source listing.

41. The order of arguments for this module as described in the documentation corresponds to the order of arguments as shown in this module's source listing.

42. The module processing as described in the documentation corresponds to the implemented processing as shown in this module's source listing.

43. The conventions (e.g., preface content, variable/module names, source code format, I/O, error handling, etc.) established in the documentation for source code development have been followed within this module.

CONSISTENCY QUESTIONS (Continued)

Internal Consistency

Note: The following questions relate to the consistency considerations within the module source listing.

44. The delineation of comments is uniform within sections of this module (e.g., within preface, within declaration section, within the body of the code).

45. Each variable in this module is considered to be of one (and only one) data type for all occurrences.

46. Each variable in this module is used for only one purpose.

47. Global variables are distinguishable from local variables by a naming convention.

48. The use of indentation is uniform within this module.

49. The information in the preface block is consistent with the associated source code (e.g., inputs, outputs, purpose, special processing, limitations, etc.).

SIMPLICITY QUESTIONS

General Coding Simplicity

Note: The following questions relate to the use of general coding techniques which affect the module simplicity.

50. The source language for this module is a high order language (HOL).
51. The control flow of this module is essentially from top to bottom.
52. This module contains very little extraneous (dead or of no functional use) code.
53. There is minimal use of specialized coding techniques (e.g., masking, bit manipulation, machine dependencies, etc.) in this module.
54. Esoteric (clever) programming is avoided in this module.
55. GOTO-like branch statements in this module are used only where essential.
56. There is minimal use of statement labels in this module.
57. A knowledge of mathematics beyond basic algebra is not required to understand the mathematical functions performed by this module.

Singular Coding Simplicity

Note: The following questions relate to the use of singular (non-compound) source language constructs.

58. This module contains a minimal number of compound (nested, non-singular) data structures.
59. This module contains a minimal number of compound (nested, non-singular) control structures.
60. Each physical source line in this module contains at most one executable source statement.
61. There is minimal use of compound Boolean expressions in this module.

Size Simplicity

Note: The following questions relate to various "counts" which reflect the amount of information which must be assimilated to understand a module.

62. The number of expressions used to control branching in this module is manageable.
63. The number of unique operators in this module is manageable.
64. The number of unique operands in this module is manageable.
65. The number of executable statements in this module is manageable.

EXPANDABILITY QUESTIONS

General Expandability

Note: The following questions relate to the generality of the module's source code which affects expansion (or contraction)/change.

66. There is minimal mixing of I/O functions and other application functions in this module.
67. There is minimal mixing of machine dependent functions and other application functions in this module.
68. Constants used repeatedly in this module are parameterized.
69. There is minimal use of processing-dependent code (e.g., relative addressing, self-modifying code, etc.) in this module.

Processing Expandability

Note: The following questions relate to the flexibility of the module code to accommodate an increase (or decrease)/change in data structure sizes, execution time, or functional requirements.

70. The size of any data structure (e.g., array) which affects the processing logic of this module is parameterized.
71. Attributes (e.g., accuracy, convergence, timing) which affect processing in this module are parameterized.
72. There is a reasonable margin between the actual execution time and the timing allocation for this module.
73. The volume of data which this module can process does not appear to be limited.
74. It appears that functional parts could be easily inserted, deleted, or replaced within this module.

INSTRUMENTATION QUESTIONS

Processing Instrumentation

Note: The following questions relate to module processing which requires particular testing considerations.

75. This module contains checks (or checks can be inserted via run-time option) for possible out-of-bound array subscripts.

76. This module contains checks to accommodate possible undefined operations (e.g., divide by zero, square root of negative number, singular matrix operation, numerical divergence, etc.).

77. This module contains a minimal amount of code which would require detailed testing (e.g., numerical algorithms, interface coordination, timing scheme, logic paths).

Control of Instrumentation

Note: The following questions relate to the presence of specific aids in the module's source code, or available through language features, which help in controlling the test/re-test of a module.

78. Source listing comments suggest or reference input data and associated output results for use in testing this module.

79. Diagnostic messages/error codes are output when an illegal input (format or range) to this module is encountered.

80. Diagnostic messages/error codes are output wherever an internal module failure could occur.

81. Intermediate results within this module can be selectively collected for eventual (or immediate) display.

82. Aids exist or can be easily inserted into the module's source code for the purpose of tracing the logical flow of control.

GENERAL QUESTIONS

Note: The following questions relate to the evaluator's general impression of the contribution the module's source listing makes to module maintainability. The definitions of maintainability and the maintainability evaluation test factors as found in the evaluator guideline handbook should be reviewed before completing these questions.

83. Modularity as reflected in this module's source listing contributes to the maintainability of this module.

84. Descriptiveness as reflected in this module's source listing contributes to the maintainability of this module.

85. Consistency as reflected in this module's source listing and between the source listing and documentation contributes to the maintainability of this module.

86. Simplicity as reflected in this module's source listing contributes to the maintainability of this module.

87. Expandability as reflected in this module's source listing contributes to the maintainability of this module.

88. Instrumentation as reflected in this module's source listing contributes to the maintainability of this module.

89. Overall it appears that the characteristics of this module's source listing contribute to the maintainability of this module.

SECTION IV
QUESTION RESPONSE
GUIDELINES

A. INSTRUCTIONS

The following sections contain information which should help clarify the intent of each question to the evaluator. Section IV.B contains information on each Software Documentation Questionnaire question. Section IV.C contains information on each Module Source Listing Questionnaire question. Section IV.D contains selected examples of documentation and code sections which should aid the evaluator in understanding some of the issues addressed by the questions.

Each question contains question terminology clarification and special response instructions as necessary.

B. SOFTWARE DOCUMENTATION QUESTIONS

Each subsection within this section corresponds to a question from the Software Documentation Questionnaire. Many questions have special response instructions which should be reviewed. Refer to section III.A for the associated question statement.

B.1 QUESTION 1

B.1.1 Terminology Clarification

part: section, volume, document, subsection, etc. as appropriate.

external interfaces: program input and output data.

B.1.2 Special Response Instructions

Answer A if one separate part exists.

Answer B-E if the external interfaces are described in several separate parts depending upon the effectiveness of that distribution.

Answer F if no description of external interfaces is available.

B.2 QUESTION 2

B.2.1 Terminology Clarification

part: section, volume, document, subsection, etc. as appropriate.

major program function: as defined by the program overview or other equivalent program information: may be a component, module, etc.

B.2.2 Special Response Instructions

Answer A if a separate part exists for each major function.

Answer B-E if each major function is described in several separate parts depending upon the effectiveness of that distribution.

Answer F if there is no description of each major function.

B.3 QUESTION 3

B.3.1 Terminology Clarification

part: section, volume, document, subsection, etc. as appropriate.

global data base: set of all variables, constants, etc. which can be accessed by more than one program module: e.g., FORTRAN, COMMONS, JOVIAL COMPOOL, ASSEMBLY DATA MODULE, etc.

B.3.2 Special Response Instructions

Answer A if a separate part exists or there is clearly no global data.

Answer B-E if the global data base is described in several separate parts depending upon the effectiveness of that distribution.

Answer F if no description of the global data base exists.

B.4 QUESTION 4

B.4.1 Terminology Clarification

major parts: as essentially defined by documentation table of contents and physical structure (volumes, sections, units, etc.): might include major functions, data base description, external interfaces, test plan, conventions and standards, etc.

self-contained: makes no cross references to other major parts of the documentation; some essential cross referencing for the purpose of eliminating bulky redundancies is acceptable.

B.4.2 Special Response Instructions

Sampling major parts of the documentation for the amount of cross referencing and the essential nature of the cross referencing should give the evaluator a general impression as to level of agreement/ disagreement with the question statement.

B.5 QUESTION 5

B.5.1 Terminology Clarification

distinct functions: these might include functional specification, detailed specification, maintenance manual, user's guide, data base description, problem reports, installation instructions, documentation plan, test plan, etc.

B.5.2 Special Response Instructions

Each (set of) volume's introduction should include an indication of what the (set of) volume's function is. A brief scan of the documents would give the evaluator a general impression of whether that function is relatively distinct, mixed, matches the stated function, etc.

The evaluator should check each physically separate volume and form an accumulative impression of the level of agreement/disagreement with the question statement.

B.6 QUESTION 6

B.6.1 Terminology Clarification

global data: any variable or constants which can be accessed by more than one module of a program.

global data structure: a particular grouping of global data variables and/or constants; e.g., FORTRAN COMMON, JOVIAL COMPOOL

B.6.2 Special Response Instructions

Documentation describing the program global data base should include the set of all global data and how it has been partitioned into global data structures.

Answer A if there is no global data (hence no global data structures); the implication is that data coupling is decreased if there is no global data.

B.7 QUESTION 7

B.7.1 Terminology Clarification

type: examples of types of data structures would be integer, real, character, array of integer, array of real, array of characters, records, files, etc.

Typical multiple use of data storage locations would be memory management schemes, equivalence and overlays. Program documentation (perhaps at the module level) should describe any use of storage locations for these types of uses.

B.7.2 Special Response Instructions

None

B.8 QUESTION 8

B.8.1 Terminology Clarification

top down hierarchical tree pattern: one imagines a root system of a tree with each junction (node) representing a major program function or module and each branch a control path between the nodes.

B.8.2 Special Response Instructions

The documentation should include a program overview section which will likely describe in narrative or chart form the overall program control flow among modules.

Control paths which are not strictly down or up in the sense of level tend to detract from modularity because of the associated lack of independence which is introduced. The evaluator's response should be accordingly (dependent upon extent) less "agreeable".

Answer F if there is no description (narrative or chart) of overall program control flows.

B.9 QUESTION 9

B.9.1 Terminology Clarification

initialization: the preparatory steps required to set the initial program data and control states.

B.9.2 Special Response Instructions

The documentation describing the program functions and control flow should also describe how the initial program state is determined (e.g., via execution of one initialization module or not). Checks of module processing may indicate whether any initialization processing is mixed with other application functions.

It is usually better if each function, module, submodule, etc. does not handle its own global initialization. There would be one part (e.g., a few modules) of the program which is for initialization. If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for initialization, then strong agreement with the question should be so indicated. If in addition, these initialization modules are all executed in preparation to any other functional activity as the first program action, then there should be essentially complete agreement with this question's statement. Variations on the program initialization processing should result in appropriate variations in the evaluator response depending upon how much initialization is mixed with other application functions.

B.10 QUESTION 10

B.10.1 Terminology Clarification

termination: the terminal steps required to set the final program data and control states (might be due to normal/abnormal termination).

B.10.2 Special Response Instructions

The documentation describing the program functions and control flow should also describe how the final program state is determined (e.g., via execution of one termination module or not). Checks of module processing may indicate whether any termination processing (e.g., FORTRAN STOP statements) is mixed with other application functions.

It is best if each function, module, submodule, etc. does not handle its own termination. There should be one part (e.g., a few modules) of the program which is for termination processing. If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for termination processing, then strong agreement with the question should be so indicated. If in addition, these termination modules are executed only as a systematic program termination procedure, then there should be essentially complete agreement with this question's statement. Variations on the program termination processing should result in appropriate variations in the evaluator response depending on how much termination processing is mixed with other application functions.

B.11 QUESTION 11

B.11.1 Terminology Clarification

I/O: input or output of program data.

B.11.2 Special Response Instructions

The documentation describing the program functions and control flow should also describe how the program I/O is done (e.g., via execution of one module or more). Checks of module processing may indicate whether any I/O functions are mixed with other application functions.

It is best if each function, module, submodule, etc. does not handle its own I/O. There should be one part (e.g., a few modules) of the program which is for I/O processing. If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for I/O, then appropriate agreement with the question should be so indicated. Variations on the program I/O processing should result in appropriate variations in the evaluator response depending on how much I/O is mixed with other application functions.

B.12 QUESTION 12

B.12.1 Terminology Clarification

error processing: the steps required to set program data and control states following the detection of an error condition.

B.12.2 Special Response Instructions

The documentation describing the program functions and control flow should also describe how the program processes any error condition (e.g. via execution of one error processing module or not). Checks of module processing may indicate whether any error processing functions are mixed with other application functions.

It is best if each function, module, submodule, etc. does not handle its own error processing unless adequate corrective measures are appropriate. There should be one part (e.g., a few modules) of the program which is for error processing. If the partitioning is such that each function is performed by a set of modules and there is one module in each set expressly for error processing, then appropriate agreement with the question should be so indicated. If in addition, these error processing modules are systematically organized as an error processing function, then there should be essentially complete agreement with this question statement.

B.13 QUESTION 13

B.13.1 Terminology Clarification

physically separate: separately bound.

B.13.2 Special Response Instructions

Answer A to F depending upon the percentage of documents which have a table of contents: e.g., 100% - answer A, 0% - answer F.

B.14 QUESTION 14

B.14.1 Terminology Clarification

physically separate: separately bound.

acronym: a term formed by the initial letter(s) of a series of one or more words.

Example: FORTRAN = FORmula TRANslator.

B.14.2 Special Response Instructions

Answer A to F depending upon the percentage of documents which have a glossary: e.g., 100% - answer A, 0% - answer F.

B.15 QUESTION 15

B.15.1 Terminology Clarification

physically separate: separately bound.

B.15.2 Special Response Instructions

Answer A to F depending upon the percentage of documents which have an index: e.g., 100% - answer A, 0% - answer F.

B.16 QUESTION 16

B.16.1 Terminology Clarification

None

B.16.2 Special Response Instructions

All documentation should be sampled in order to get a general impression as to the ease of locating information. The evaluator is encouraged to repeatedly conceptualize the need for locating some maintenance type of information and then checking the documentation for the effort required to locate the information.

B.17 QUESTION 17

B.17.1 Terminology Clarification

version description document: a document which describes the version of each computer program configured item of the software.

current: up to date, useful.

B.17.2 Special Response Instructions

None

B.18 QUESTION 18

B.18.1 Terminology Clarification

master list: this may be a reference list in one overview document, or a separate document itself.

software documentation: the list should contain at least all delivered program-related documentation by name and description; if several programs are part of the software development effort, then this list may contain information on all programs.

B.18.2 Special Response Instructions

None

B.19 QUESTION 19

B.19.1 Terminology Clarification

dynamic allocation: any assignment of a resource which is (or can be) done during the execution of a program; contrasts with "static allocation" which implies the resource assignment remains fixed throughout program execution.

explain: to make easily understandable.

The documentation describing the functional/detailed program specifications should include a section which explains what dynamic allocation features are used by the program. The most normal dynamic allocation feature is probably storage allocation. There may be allocation routines which must be called to get or release memory. Also, the priority scheme or timing allocation for particular "rate" groups may depend upon certain phases of a mission and dynamically change on that basis. Likewise assignment of control of tape drives, discs, communication lines or other hardware may be done in some dynamic manner. Many of these features may be considered necessary depending upon the application, but all are considered to increase the effort required to maintain a program.

B.19.2 Special Response Instructions

Answer A if it appears there is no dynamic allocation of resources for this program.

Answer B-F otherwise.

B.20 QUESTION 20

B.20.1 Terminology Clarification

timing requirements: these requirements include the actual allocated time for each major function as well as the timing relationship (scheme) among the major functions.

major function: the program overview, hierarchical chart, etc. will ordinarily define what major function (and its components) means; it usually will correspond to a module or group of modules as defined for a given program evaluation.

explain: to make easily understandable.

B.20.2 Special Response Instructions

Answer A if it appears that this program has no timing requirements (e.g., is non-real time).

Answer B-F otherwise.

B.21 QUESTION 21

B.21.1 Terminology Clarification

storage requirements: core storage allocation.

major function: the program overview, hierarchical chart, etc. will ordinarily define what major function (and its components) means; it usually will correspond to a module or group of modules as defined for a given program evaluation.

explain: to make easily understandable.

B.21.2 Special Response Instructions

Answer F if there is no explanation of the core storage allocation of the program. Note: Even if a program does not have any "critical" storage requirements (e.g., program executing on large computer), there should be an explanation in the documentation concerning the program's environment (which would include some explanation of storage capacity, requirements, etc.).

B.22 QUESTION 22

B.22.1 Terminology Clarification

inputs: global data and other parameters passed via an argument list which are used by the module.

explain: to make easily understandable.

B.22.2 Special Response Instructions

None

B.23 QUESTION 23

B.23.1 Terminology Clarification

processing: the general algorithm necessary to generate the necessary outputs from the inputs.

explain: to make easily understandable.

B.23.2 Special Response Instructions

None

B.24 QUESTION 24

B.24.1 Terminology Clarification

outputs: global data and other parameters passed via an argument list which are assigned a value within the module.

explain: to make easily understandable.

B.24.2 Special Response Instructions

None

B.25 QUESTION 25

B.25.1 Terminology Clarification

explain: to make easily understandable.

B.25.2 Special Response Instructions

None

B.26 QUESTION 26

B.26.1 Terminology Clarification

flowchart (or equivalent): FORTRAN flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO) chart, etc.

diagramatically illustrates: makes clearly understood in a basically non-narrative manner.

B.26.2 Special Response Instructions

Answer F if there is no flowchart (or equivalent).

B.27 QUESTION 27

B.27.1 Terminology Clarification

initialization: the preparatory steps required to set the initial program data and control states.

termination: the terminal steps required to set the final program data and control states (might be due to normal/abnormal termination).

explain: to make easily understood.

B.27.2 Special Response Instructions

The evaluator should study both initialization and termination processing explanations. The response A-F should reflect overall how well both have been explained.

B.28 QUESTION 28

B.28.1 Terminology Clarification

recovery: the procedures taken to report/correct some program failure (resulting from an external error condition in this case and probably recognized as bad input data).

explain: to make easily understandable.

B.28.2 Special Response Instructions

The documentation describing the major program functions should include an explanation of overall error processing including the recovery from externally generated error conditions. Most likely this explanation will be of recovery from bad/no program input data.

B.29 QUESTION 29

B.29.1 Terminology Clarification

process of recovering: the procedures taken to report/correct some recognized internal error condition.

internal error condition: any algorithm failure due to processing of internally defined data.

explain: to make easily understandable.

B.29.2 Special Response Instructions

The documentation describing the major program functions should include an explanation of overall error processing including the recovery from program failures resulting from an internal error condition.

B.30 QUESTION 30

B.30.1 Terminology Clarification

input: what data is input and how the input is accomplished.

explain: to make easily understood.

B.30.2 Special Response Instructions

None

B.31 QUESTION 31

B.31.1 Terminology Clarification

output: what data is output and how the output is accomplished.

explain: to make easily understood.

B.31.2 Special Response Instructions

None

B.32 QUESTION 32

B.32.1 Terminology Clarification

chart: flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO) chart, etc.

B.32.2 Special Response Instructions

The set of charts may combine data and control flow or may show the data flow and control flow on separate charts. Either of these methods, as well as a general narrative description, could be satisfactory. The criterion is how understandable and complete such charts (descriptions) appear to the evaluator.

Answer F if no set of charts exists.

B.33 QUESTION 33

B.33.1 Terminology Clarification

global variable: any variable which can be accessed by more than one module of a program; global constants should also be identified.

B.33.2 Special Response Instructions

The table of contents of the functional/detailed program specification should indicate whether such a cross reference exists. Occasionally there will be an automated program cross reference list which serves this purpose.

Answer F if no master list or its equivalent exists.

B.34 QUESTION 34

B.34.1 Terminology Clarification

global variable: any variable which can be accessed by more than one module of a program; global constants should also be identified.

B.34.2 Special Response Instructions

Answer F if no master list or its equivalent exists.

B.35 QUESTION 35

B.35.1 Terminology Clarification

complex mathematical model: e.g., Fourier transform, Laplace transform, numerical integration/differentiation scheme, control theory algorithm, statistical technique, etc.

explain: to make easily understandable.

B.35.2 Special Response Instructions

Answer A if it is clear that there are no complex mathematical models (techniques, algorithms) used in the program.

B.36 QUESTION 36

B.36.1 Terminology Clarification

complex mathematical model: e.g., Fourier transform, Laplace transform, numerical integration/differentiation scheme, control theory algorithm, statistical technique, etc.

B.36.2 Special Response Instructions

Answer A if it is clear that there are no (complex) mathematical models (techniques, algorithms) used in the program.

B.37 QUESTION 37

B.37.1 Terminology Clarification

standards: either local contractor standards or universal standards (e.g., ANSI FORTRAN) which help understandability; usually the format consistency of the documentation indicates how much a standard/convention, etc. has been followed.

B.37.2 Special Response Instructions

None

B.38 QUESTION 38

B.38.1 Terminology Clarification

standards: either local contractor standards or universal standards (e.g., ANSI FORTRAN) which help understandability; usually, the format consistency of the flowchart indicates how much a standard/convention, etc. has been followed.

flowchart (or equivalent): FORTRAN flowchart, Process Design Language (PDL), Hierarchical Input-Processing-Output (HIPO), etc.

B.38.2 Special Response Instructions

The flowcharts of the program and modules should be scanned as to conventions for symbols, labeling consistency, etc. There may be a stated standard (e.g., ANSI FORTRAN, etc.) which may be used as a guideline to compare against the flowcharts.

Answer F if there are no flowcharts (or equivalents).

B.39 QUESTION 39

B.39.1 Terminology Clarification

major functional part: the program overview, hierarchical chart, etc. will ordinarily define what major function (and its components) means; it usually will correspond to a module or group of modules as defined for a given program evaluation.

same format: each major functional part typically have at least a functional overview with a description of each component, module, etc. and the inputs processing and outputs of each component, module, etc.

B.39.2 Special Response Instructions

None

B.40 QUESTION 40

B.40.1 Terminology Clarification

format of the documentation: documents, volumes, sections, etc.

organization of the program: design of the program as components, modules, global data base, units segments, etc.

B.40.2 Special Response Instructions

The idea behind this question is that the program parts as designed (and likely implemented) are easier to maintain if the documentation organization (format), by including sections for the description of those designed parts, makes it easy to locate details concerning those program parts.

For example major program functions, the program data base, etc. might have separate sections. As another example, the descriptions of the how the program is designed to be tested should be reflected in the format of the documentation such as providing sections for unit test procedures and sample test data (input and output), if appropriate.

There are many other considerations which may influence the evaluator in responding to this question. What is desired is basically the evaluator's general impression as to the usefulness of the documentation format in understanding the overall program organization.

B.41 QUESTION 41

B.41.1 Terminology Clarification

None

B.41.2 Special Response Instructions

There should be documentation which identifies program design conventions. In addition, the module descriptions can be scanned to determine whether such conventions have been established and/or followed. The establishment of prologue and epilogue syntax conventions is especially important for assembly modules.

B.42 QUESTION 42

B.42.1 Terminology Clarification

I/O processing: the physical input or output of program data.

B.42.2 Special Response Instructions

There should be documentation which identifies program design conventions. In addition, the module descriptions can be scanned to determine if any particular design consistency/conventions have been considered for program I/O processing.

B.43 QUESTION 43

B.43.1 Terminology Clarification

error processing: the procedure followed after a program failure due to some recognized error condition.

B.43.2 Special Response Instructions

There should be documentation which identifies program design conventions. In addition, the module descriptions can be scanned to determine if any particular design consistency/conventions have been considered for program error processing.

B.44 QUESTION 44

B.44.1 Terminology Clarification

naming convention: for example, in FORTRAN all module names will have six alphanumeric characters ending in X, and each name shall be an acronym for a long name descriptor. Or, the first two characters of each module will be an acronym for the component (group) to which the module belongs.

B.44.2 Special Response Instructions

Whether a naming convention for modules has been used should be apparent by observing the actual module names and/or scanning the descriptions of each module.

The response should reflect how useful the naming of the modules is for purposes of recognizing the module function, where it belongs in the overall program organization, etc. Usually, a naming convention will help fulfill this purpose.

B.45 QUESTION 45

B.45.1 Terminology Clarification

naming convention: for example, all global variables shall be exactly six alphanumeric characters long, and non-global variables shall be five or fewer characters long.

global variable: any variable which can be accessed by more than one module of a program; global constants should also follow the naming convention.

B.45.2 Special Response Instructions

The documentation should identify any naming conventions in a separate "conventions and standards" part. In addition, if global variables are used there should be a description of the global data base. By scanning the global variable names it should be apparent whether any naming convention has been established which might aid in identifying which variables as used in a module are/are not global.

The response should reflect how useful the naming convention is for the purpose of recognizing the function of the global data (variable or constant) and in distinguishing the global data from any local data.

Answer A if there is no global data.

B.46 QUESTION 46

B.46.1 Terminology Clarification

terminology: the general use of English and program terms should be simple, straightforward, easily understood; any terms needing to be clarified should be defined prior to use and included in a glossary for reference.

B.46.2 Special Response Instructions

None

B.47 QUESTION 47

B.47.1 Terminology Clarification

physically organized: the documents, volumes, chapters, sections, etc.

B.47.2 Special Response Instructions

Generally, the documentation should reflect by the products produced during a software development effort - requirements, preliminary design, detailed design, operation/maintenance manual, test plan, etc. This will reflect a natural progression of program description from levels of less detail to levels of more detail. In addition, within any given documentation product, e.g., the detailed design specification, there should be a sequential progression from descriptions of less detail (e.g., overview) to descriptions of more detail (e.g., module design).

B.48 QUESTION 48

B.48.1 Terminology Clarification

concern: idea, topic, etc.

B.48.2 Special Response Instructions

All program documentation should be scanned. If the documentation has been written in a simple understandable manner then more than likely each part will address one primary topic (and subparts, one primary subtopic, etc.). The descriptions will be simple and to the point.

B.49 QUESTION 49

B.49.1 Terminology Clarification

amount: number of cross references as well as the method of indicating such references.

cross reference: a note, statement, section number, etc. which directs a reader from one part of the documentation to another part.

contributes: too much cross referencing or in certain instances too little cross referencing can make the program description difficult to follow.

B.49.2 Special Response Instructions

Some parts of the documentation may use cross referencing well while other parts may not. The evaluator should study the documentation until a reasonably well-founded overall impression is formed.

B.50 QUESTION 50

B.50.1 Terminology Clarification

high order language: e.g., non-assembly, non-micro code, non-machine, FORTRAN, JOVIAL, PL/I, PASCAL, etc.

B.50.2 Special Response Instructions

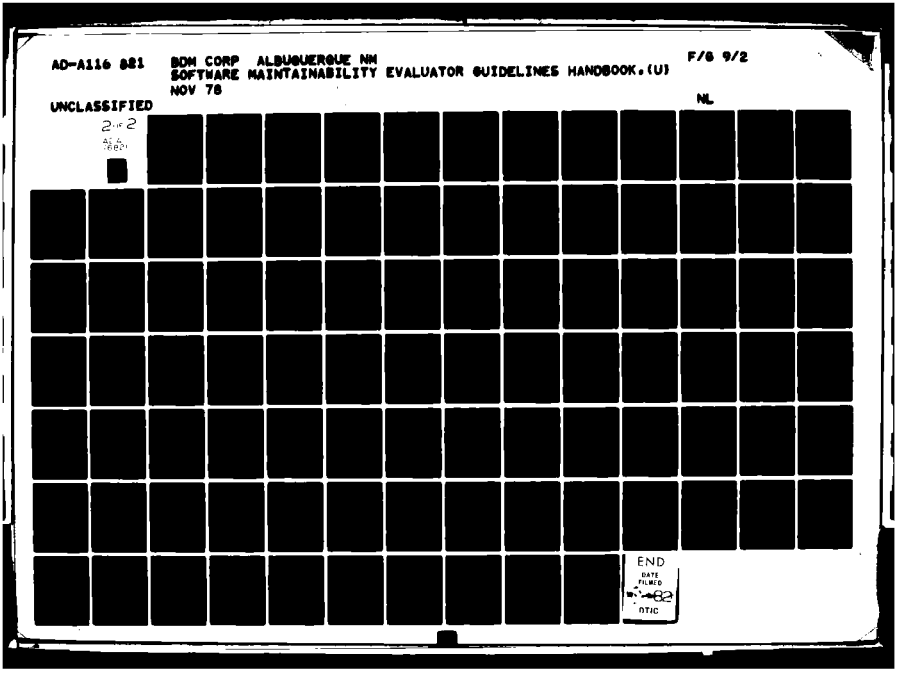
Answer A if the program source language is completely HOL. Answer F if the program source language is completely non-HOL. Answer in the range B to E if the source language is a mix of HOL and non-HOL by approximate percentage:

B - $\geq$ 80%

C - $\geq$ 60%

D - $\geq$ 40%

E - $\geq$ 20%


AD-A116 881
BDM CORP ALBUQUERQUE NM
SOFTWARE MAINTAINABILITY EVALUATOR GUIDELINES HANDBOOK.(U)
NOV 78
F/G 9/2
UNCLASSIFIED
NL
2 OF 2
END
DATE FILMED
DTIC

B.51 QUESTION 51

B.51.1 Terminology Clarification

recursive programming technique: the use of operations which are defined in terms of themselves: a recursive module is one which uses a call to itself within the body of the code.

reentrant programming technique: the technique of interrupting a program module at any point by another user and then resuming execution at the point of interruption. A reentrant module is one which can be concurrently used by more than one user.

excessive: detracts from simplicity.

B.51.2 Special Response Instructions

The documentation should identify within a general "program design considerations" section or the individual module description sections whether recursion or reentrancy is to be utilized. Many languages (or at least a particular implementation of a compiler) do not allow recursive and/or reentrant code. Some languages (e.g., stack-oriented languages like Pascal) allow recursion as a natural language capability.

The evaluator should get an overall impression of how much recursion/reentrant programming is a part of the overall program design. If done with care some use of recursion or reentrancy can simplify the overall program design even though the particular modules which are recursive/reentrant will probably be harder to maintain because of those concerns. Utility modules generally fall into this class of modules.

B.52 QUESTION 52

B.52.1 Terminology Clarification

None

B.52.2 Special Response Instructions

The evaluator should either agree or disagree with the question statement on the basis of the overall extent to which the program modules satisfy the requisite of one function per module.

B.53 QUESTION 53

B.53.1 Terminology Clarification

resource allocation: the assignment of a particular resource to a particular program task, function, module, etc.

fixed: is not re-assigned from initialization to termination or reinitialization of the program.

B.53.2 Special Response Instructions

The sharing or dynamic re-assignment of resources should be a highlight of a section describing special processing (control) considerations, memory allocation, timing requirements by mission phase, etc. As another recourse, the evaluator can check the individual module descriptions for possible mention of any dynamic resource allocation.

B.54 QUESTION 54

B.54.1 Terminology Clarification

control flow among modules: which modules call and are called by other modules.

B.54.2 Special Response Instructions

The documentation should include narrative or a hierarchical flowchart which gives a general overview of the sequence in which modules (and perhaps submodules) are invoked and what controls that sequence.

B.55 QUESTION 55

B.55.1 Terminology Clarification

timing scheme: time slicing, time sharing, priority levels, rate groups, etc. as applied to the overall sequencing and execution of program functions.

B.55.2 Special Response Instructions

The program documentation should include a separate section which describes overall program timing requirements and the timing scheme designed to satisfy those requirements.

Answer A if there is no timing scheme because the program requires no special timing considerations (e.g., is non-real time).

B.56 QUESTION 56

B.56.1 Terminology Clarification

interrupted: execution is suspended without the knowledge of the module being suspended.

B.56.2 Special Response Instructions

None

B.57 QUESTION 57

B.57.1 Terminology Clarification

basic algebra: functions, equations, polynomials, graphing of functions, basic manipulations, etc.; excludes calculus, differential, equations, Fourier transforms, statistical algorithms, etc.

B.57.2 Special Response Instructions

The evaluator should respond on the basis of overall program considerations. There may be a few complex functions, but on the average most of the functions require no mathematics beyond basic algebra. In this case, the evaluator might generally or strongly agree with the question statement. If there appear to be many complex functions, the evaluator may want to generally or strongly disagree with the question statement.

B.58 QUESTION 58

B.58.1 Terminology Clarification

numbering scheme: volume numbering, section/subsection/paragraph numbering, page numbering. Example: Consecutive page numbering makes it difficult to add/delete or even change pages without changing the numbers of succeeding pages. Numbering pages by section is a better technique.

B.58.2 Special Response Instructions

The documentation organization into sections, subsections, etc. should be scanned. Also, the page numbering should be checked.

B.59 QUESTION 59

B.59.1 Terminology Clarification

None

B.59.2 Special Response Instructions

All program documentation should be scanned for graphic materials and the manner in which these materials are separated from narrative description.

B.60 QUESTION 60

B.60.1 Terminology Clarification

numbering scheme: Example: Consecutive numbering of figures across major sections makes it more difficult to add or delete figures than numbering consecutively within a major section.

B.60.2 Special Response Instructions

All program documentation should be scanned for graphic materials and the manner in which these materials have been numbered.

B.61 QUESTION 61

B.61.1 Terminology Clarification

timing scheme: time slicing, time sharing, priority levels, rate groups, etc. as applied to the overall sequencing and execution of program functions.

flexible: Could one function be moved from one priority level to another? Is the organization of each rate group (collection of functions which execute every "n" time units) designed to near capacity? What is necessary in order that a function be added to deleted from or moved among rate groups? Is the time sharing/slicing scheme interrupt driven, based on a static allocation, etc.? Do the functions have some specific interface with the time sharing controller or is the function definition essentially independent of specific time sharing considerations?

B.61.2 Special Response Instructions

Answer A if there are no required program timing considerations (e.g., program is non-real time). Answer B-F otherwise.

B.62 QUESTION 62

B.62.1 Terminology Clarification

program function: this term is used generically to mean whatever is appropriate for the particular application as far as timing considerations are concerned; it may not actually correspond to a defined program function but may be an accumulation of several functions (or parts of functions) into a rate group or a priority level, etc.

B.62.2 Special Response Instructions

Answer A if the program has no timing requirements and hence timing margins are of no concern (e.g., program is non-real time). Answer A if each timing margin is at least 25%. Answer B-F otherwise.

B.63 QUESTION 63

B.63.1 Terminology Clarification

explain: to make easily understandable.

basic data storage sizes: the size of program data structures upon which program processing depends; it may be array sizes, storage allocated by an assembly directive, etc.

B.63.2 Special Response Instructions

None

B.64 QUESTION 64

B.64.1 Terminology Clarification

None

B.64.2 Special Response Instructions

Answer A if at least 25% storage increase is available. Answer B-F otherwise.

B.65 QUESTION 65

B.65.1 Terminology Clarification

data structure: e.g., array, FORTRAN COMMON, etc.

B.65.2 Special Response Instructions

Answer A if it is clear that no modules are dependent upon data structure sizes. This situation is probably not likely in large software programs.

B.66 QUESTION 66

B.66.1 Terminology Clarification

functional parts: primarily modules, but also includes submodules and groups of modules into major functions or components.

B.66.2 Special Response Instructions

The evaluator should form an impression from the module descriptions and program overview information whether the functional parts could be added or deleted. For example, functions that one executed as a result of a table driven executive can generally be easily replaced or deleted, and new functions are easily added to the table. The more intricate the interfaces, the less likely modules or other functions can easily be added or deleted.

B.67 QUESTION 67

B.67.1 Terminology Clarification

part: section, volume, document, subsection, etc. as appropriate.

program test plan: set of descriptions and procedures for how the program is to be (or can be, or has been) tested.

B.67.2 Special Response Instructions

Answer A if a separate part exists. Answer F if a separate part does not exist. If for some reason the program test plan description is distributed over several separate parts (e.g., one part per unit/module description), then answer in the range B to E as to the effectiveness of that "separation" from the point of view of program test/retest.

B.68 QUESTION 68

B.68.1 Terminology Clarification

part: section, volume, document, subsection, etc. as appropriate.

sample test data: the input and output data used for the program tests.

B.68.2 Special Response Instructions

Answer A if a separate part exists. Answer F if a separate part does not exist. If for some reason the sample test data description is distributed over several separate parts (e.g., one part per unit/module description), then answer in the range B to E as to the effectiveness of that "separation" from the point of view of program test/retest.

B.69 QUESTION 69

B.69.1 Terminology Clarification

part: section, volume, document, subsection, etc.

program support tools: general debug aids, test/retest software, trace software/hardware features, use of compiler/link editor/ library management/configuration management/test editor/display software tools.

B.69.2 Special Response Instructions

Answer A if a separate part exists. Answer F if a separate part does not exist. If for some reason the description of program support tools for testing is distributed over several separate parts then answer in the range B to E as to the effectiveness of that "separation" from the point of view of program test/retest.

B.70 QUESTION 70

B.70.1 Terminology Clarification

None

B.70.2 Special Response Instructions

The program test plan should either include the set of test procedures or reference such a set. The maintenance/operational documentation might also be a source for such test procedures.

One good test of the explanation of the program test procedures is for the evaluator to visualize how easy it would be to execute step by step one or more of the particular test procedures. If the information is not presented in a step by step fashion with a complete discussion of the test environment, test inputs and expected test outputs, then the test will probably be difficult to perform.

If no test procedures are available, then answer F.

B.71 QUESTION 71

B.71.1 Terminology Clarification

unit: the test procedures will ordinarily be described in terms of unit testing information and integration testing information, where the units may be modules, submodules, groups of modules or some other organization depending upon the contractor and the application area.

B.71.2 Special Response Instructions

The program test plan should either include the set of test procedures or reference such a set. The maintenance/operational documentation might also be a source for such unit test information. Another source might be the individual sections describing major program functions/modules.

One good test of the usefulness of the unit test information is for the evaluator to visualize how easy it would be to execute step by step fashion with a complete discussion of the test environment, test inputs and expected test outputs, then the test will probably be difficult to perform.

B.72 QUESTION 72

B.72.1 Terminology Clarification

None

B.72.2 Special Response Instructions

The program test plan, maintenance/operational documentation, or individual sections describing major program functions/modules would be the most likely sources for the limitations/incompleteness of the test procedures.

If no test procedures exist or there is no information on the limitations/incompleteness of the test procedures, then answer F.

B.73 QUESTION 73

B.73.1 Terminology Clarification

display: print, console display, hard copy, etc. in a clearly understandable form.

B.73.2 Special Response Instructions

None

B.74 Question 74

B.74.1 Terminology Clarification

standardized set: data which can be used to compare in a predictable manner over time program test procedure results; the set may be separated on the basis of test procedure, test unit or some other appropriate manner.

B.74.2 Special Response Instructions

None

B.75 QUESTION 75

B.75.1 Terminology Clarification

include: are in-line or can be inserted in-line.

test probe: section of code or special module which collects certain process parameters; generally the activation of the probe can be controlled through user options.

processing performance: timing, sequencing, process trace, data structure, etc. information.

B.75.2 Special Response Instructions

It should be obvious from program test information or perhaps program error processing whether such probes have been a part of the overall program design considerations. The capability to have such probes designed into the program is greatly affected by the particular source language (and associated implementation) being used.

For example, if the language provides debug capabilities or such options as conditional compilation, then the designer is much more likely to consider the use of test probes as a normal part of the program. However, it is still possible under more adverse conditions for the design to include separate functions which can be individually invoked for the purpose of collecting appropriate processing performance.

B.76 QUESTION 76

B.76.1 Terminology Clarification

None

B.76.2 Special Response Instructions

The documentation describing error processing/error codes/error messages, or perhaps report generation could be checked to determine what type of error checking appears to be done. In addition, the general design conventions/standards might indicate what error checking conventions have been adopted. The source language compiler may have options which allow for the generation of run-time parameter checking (e.g., index range).

B.77 QUESTIONS 77-83

B.77.1 Terminology Clarification

See evaluator guideline handbook.

B.77.2 Special Response Instructions

The evaluator should not average any previous responses to obtain a response to the general questions.

For each specific maintainability test factor (modularity, descriptiveness, consistency, simplicity, expandability and instrumentation) the documentation questionnaire has addressed some associated documentation characteristics. There may be more important characteristics for the particular application or the manner in which the included questions were asked may not have allowed the evaluator to express a most appropriate answer. The general questions should be answered with respect to what the evaluator's impression was concerning the extent to which each specific test factor contributes to maintainability.

And, independent of how maintainability has been partitioned into test factors, the evaluator should give a response as to the extent to which the program's documentation characteristics will contribute to overall program maintainability.

C. MODULE SOURCE LISTING QUESTIONS

Each subsection within this section corresponds to a question from the Module Source Listing Questionnaire. Many questions have special response instructions which should be reviewed. Refer to section III.B for the associated question statement.

C.1 QUESTION 1

C.1.1 Terminology Clarification

data elements: variables, constants

data structure: group of data elements and/or other data structures; e.g., array, record, file, etc.

C.1.2 Special Response Instructions

As illustrated in figure IV-1, there may be a physical grouping of functionally related data even though it is somewhat unsatisfactory in showing what the functional relationship is. The evaluator should give a response based upon how easy it is to determine the functional purpose of the data by observing how the data has been organized, grouped, etc. It is not likely that the evaluator's answer will be A or F since most modules will have some logical organization of data, but few will have a superior such organization.

As an example of data structuring, suppose a state vector for an object in track has the information: id, position, velocity, acceleration. And, suppose a maximum of 100 objects could be in track at one time. Then figure IV-1 illustrates typical data structures which might be set up for ASSEMBLY, FORTRAN, and PASCAL. As one can easily see, the data structuring capabilities of PASCAL allowed the functional relationships to be more clearly represented. FORTRAN with its array structuring capabilities helps. In ASSEMBLY, a block of storage must be managed internally.

C.2 QUESTION 2

C.2.1 Terminology Clarification

Structured programming control structures: the basic structures are SEQUENCE, DECISION and ITERATION as illustrated in figure IV-2.

C.2.2 Special Response Instructions

The concepts of structured programming control structures can be adopted whether the language is high order or assembly. Some high order language syntax guarantees that only the basic control structures can be used. Other high order languages (e.g., FORTRAN) give some help in

ASSEMBLY

```
    DS 1000                 ? DEFINE STORAGE FOR AN
                            ? ARRAY OF STATE VECTORS
                            ? EACH STATE VECTOR IS OF SIZE 10
                            ? WITH A MAXIMUM OF 100
```

FORTRAN

```
    REAL STATE (100,10), POS (100,3), VEL (100,3), ACC (100,3)
    INTEGER ID (100)
    EQUIVALENCE (STATE (1,1), ID (1)),
                (STATE (1,2), POS (1,1)),
                (STATE (1,5), VEL (1,1)),
                (STATE (1,8), ACC (1,1))
```

PASCAL

```
    CONST     NOBJ = 100;
    TYPE      R3VEC = ARRAY [1..3] OF REAL;
              OBJREC = RECORD ID : INTEGER;
                              POS : R3VEC;
                              VEL : R3VEC;
                              ACC : R3VEC
                       END;
    VAR       STATE : ARRAY [1..NOBJ] OF OBJREC;
```

Figure IV-1. Example of Data Structuring

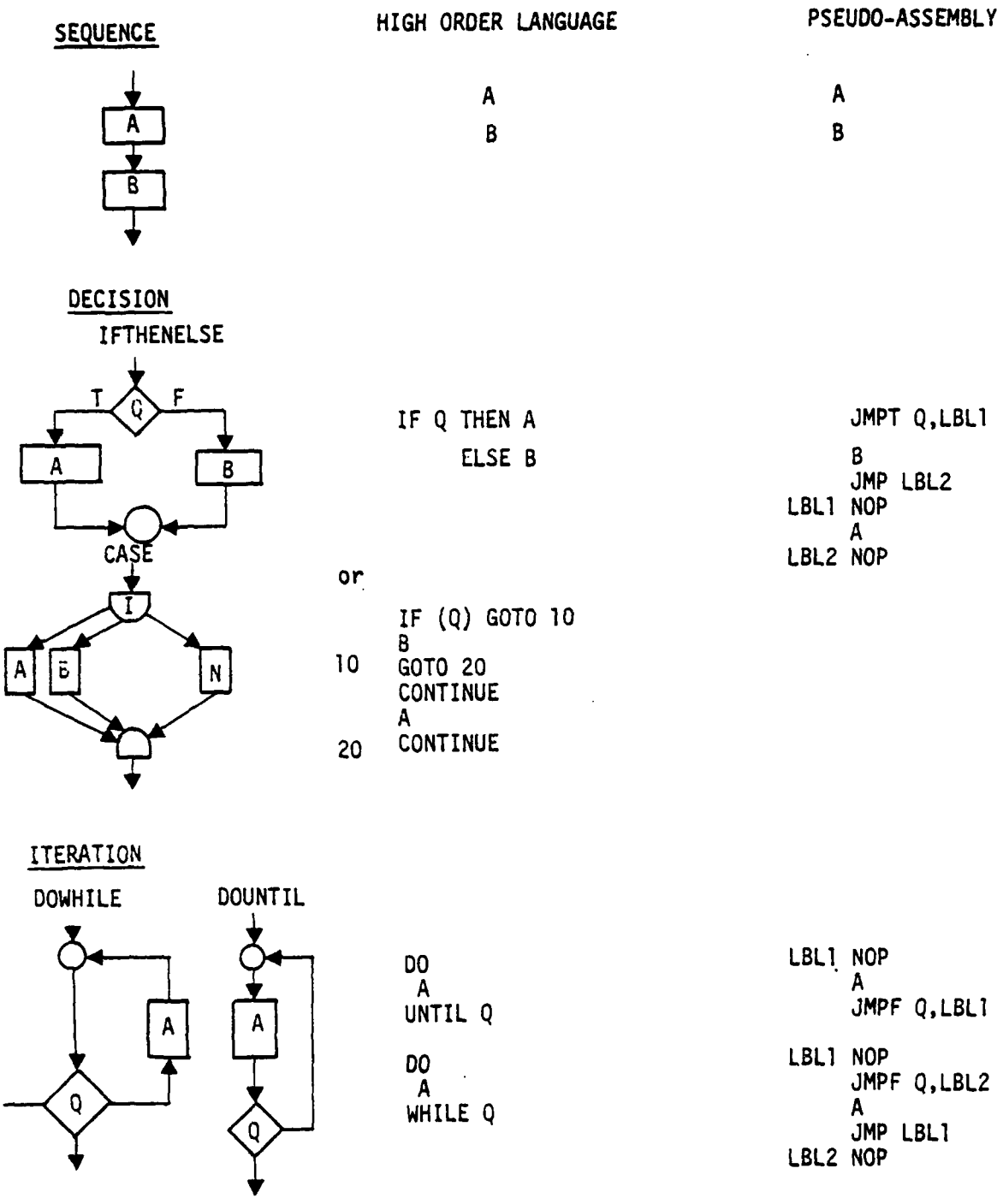


Figure IV-2. Basic Control Structures

constructing the basic control structures with judicious use of the GOTO statement. Assembly languages generally require that a convention be established in order that only the basic control structures will be consistently used. The evaluator should form a general opinion as to how well this module has conformed to the use of only these basic control structures. Answer A only if there is no deviation, answer F only if the module's logic structure is extremely disorganized.

C.3 QUESTION 3

C.3.1 Terminology Clarification

None.

C.3.2 Special Response Instructions

Answer A only if there is no data sharing of memory locations for the data used by this module.

C.4 QUESTION 4

C.4.1 Terminology Clarification

global data: variables or constants which can be accessed by more than one module.

C.4.2 Special Response Instructions

Answer A only if there is no global data used in this module.

C.5 QUESTION 5

C.5.1 Terminology Clarification

entry point: statement (address) to which control is transferred; first executable statement.

C.5.2 Special Response Instructions

Answer A only if the number of entry points is one.

C.6 QUESTION 6

C.6.1 Terminology Clarification

exit point: statement (address) from which control is transferred; last executable statement; e.g., multiple RETURN statements in a FORTRAN program reflect multiple exit points.

C.6.2 Special Response Instructions

Answer A only if the number of exit points is one.

C.7 QUESTION 7

C.7.1 Terminology Clarification

functional task: part, element; a series of contiguous computations, which produce one or more results.

related: tasks are related if the cumulative computational results are related in a logical, temporal, communicational, sequential, or functional manner and not just by coincidence. The strength of the relationship increases from logical to functional. See table IV-1.

C.7.2 Special Response Instructions

A mixture of the types of binding as given in table IV-1 will frequently be found in a module, but the range of binding from coincidental to functional is a reasonable match of the response scale from F to A, respectively.

C.8 QUESTION 8

C.8.1 Terminology Clarification

functional task: part, element; a series of computations which produce one or more results.

block of code: contiguous set of statements.

C.8.2 Special Response Instructions

None.

TABLE IV-1. Types of Module Binding

coincidental binding:	the module is created to consolidate "duplicate coding" in other modules or to adhere to a set size requirement.
logical binding:	the module does all input and output operations, or the module edits all data of several classes of data.
temporal binding:	same as logical binding plus the module is executed at a particular time; initialization, termination, clean-up, etc.
communicational:	the modules tasks are related by reference to the same set of input and/or output data of the module.
sequential binding:	the output of each task is the input for the "next" task of the module.
functional binding:	all the modules tasks are related to the performance of one well-defined function.

C.9 QUESTION 9

C.9.1 Terminology Clarification

iteration block: iteration control structure (see question 2).

entry point: statement (address) to which control is transferred; first executable statement.

C.9.2 Special Response Instructions

The intent is that there should be no jumps into the body of any iteration block. Answer A only if all iteration blocks satisfy the one entry point criterion.

C.10 QUESTION 10

C.10.1 Terminology Clarification

iteration block: iteration control structure (see question 2).

exit point: statement (address) from which control is transferred; last executable statement.

C.10.2 Special Response Instructions

The intent is that there should be no jumps out of the body of any iteration block. It is also best (but might only be a minor detraction) if there are no calls to other modules within an iteration block and that the only exit is through the terminal statement which is a natural part of the iteration block. The ESCAPE (jump from the body to the immediately following executable statement) is also only a minor detraction if used properly and not excessively. The evaluator should answer A only if there are no module calls within an iteration block and the exit is always through the related decision statement.

C.11 QUESTION 11

C.11.1 Terminology Clarification

decision block: decision control structure (see question 2).

entry point: statement (address) to which control is transferred; first executable statement.

C.11.2 Special Response Instructions

The intent is that there should be no jumps into the body of any decision block. Answer A only if all decision blocks satisfy the one entry point criterion.

C.12 QUESTION 12

C.12.1 Terminology Clarification

decision block: decision control structure (see question 2).

exit point: statement (address) from which control is transferred; last executable statement.

C.12.2 Special Response Instructions

The intent is that there should be no jumps out of the body of a decision block except as the natural part of the decision syntactic structure (see question 2). It is also best (but might only be a minor detraction) if there are no calls to other modules within a decision block. The ESCAPE (jump from the body to the immediately following executable statement) is also only a minor detraction if used properly and not excessively. The evaluator should answer A only if there are no module calls within a decision block and the exit is always through the natural decision block exit.

C.13 QUESTION 13

C.13.1 Terminology Clarification

None

C.13.2 Special Response Instructions

Answer A if there is no calling module (e.g., program module) or this module does return to the calling module. Answer F if this module does not return control to the calling module.

C.14 QUESTION 14

C.14.1 Terminology Clarification

input: global data or data parameter in argument list used in the module.

output: global data or data parameter in argument list and assigned a value within the module.

C.14.2 Special Response Instructions

Answer A only if no variables are both input and output.

C.15 QUESTION 15

C.15.1 Terminology Clarification

input: global data or data parameter in argument list used in the module.

described: includes details such as type, function, input/output, units, range, etc.

preface block: separate set of contiguous comments which precedes the main body of the module (a similar block at any other common module location would be satisfactory).

C.15.2 Special Response Instructions

If there is no preface block answer D, E, or F on the basis of whether the module inputs are clearly described at any other place in the module. More than likely the answer should be F if there is no preface block.

If there is a description of inputs in a preface block, then keep in mind that the response anchors A and F are only for extremely good or extremely poor descriptions, respectively.

C.16 QUESTION 16

C.16.1 Terminology Clarification

output: global data and data parameters in argument list which are assigned a value within the module.

described: includes details such as type, function, input/output, units, range, etc.

preface block: see question 15.

C.16.2 Special Response Instructions

If there is no preface block answer D, E, or F on the basis of whether the module outputs are clearly described at any other place in the module. More than likely the answer should be F if there is no preface block.

If there is a description of outputs in a preface block, then keep in mind that the response anchors A and F are only for extremely good or extremely poor descriptions, respectively.

C.17 QUESTION 17

C.17.1 Terminology Clarification

purpose: functions, tasks, abstract, summary, etc.

described: includes details such as basic processing of inputs to obtain outputs.

preface block: see question 15.

C.17.2 Special Response Instructions

If there is no preface block answer D, E, or F on the basis of whether the module purpose is clearly described at any other place in the module. Since it is quite usual to include embedded comments which clearly describe the module's purpose independent of whether a preface block is present, it would be unusual to respond F to this question statement even if there is no preface block.

If there is a description of the purpose in a preface block, then keep in mind that the response anchors A and F are only for extremely good or extremely poor descriptions, respectively.

C.18 QUESTION 18

C.18.1 Terminology Clarification

identified: listed.

preface block: see question 15.

C.18.2 Special Response Instructions

It is difficult to determine which modules call a given module from a source listing alone. And, it is difficult to determine whether modules listed are all the modules which call the given module. Again, if there is no preface block answer D, E, or F on the basis of whether the modules which call this module are clearly described at any place in the module.

C.19 QUESTION 19

C.19.1 Terminology Clarification

identified: listed.

preface block: see question 15.

C.19.2 Special Response Instructions

It is possible to determine which modules are called by the given module from a cursory look at the source listing. If there is no preface block, then answer D, E, or F on the basis of whether the modules which are called are clearly identified at any other place in the module. If there is a preface block and all the called modules are identified then answer A; if there are no called modules it should be obvious or so stated.

C.20 QUESTION 20

C.20.1 Terminology Clarification

described: includes details as to what the limitation is and how it is handled.

preface block: see question 15.

C.20.2 Special Response Instructions

If there is no preface block answer D, E, or F on the basis of whether the module limitations are clearly described at any other place in the module. Answer A if there is a preface block and it is either obvious or so stated that there are no limitations which need a description (check the source listings to verify).

If there is a description of limitations in a preface block, then keep in mind that the response anchors of A and F are only for extremely good or extremely poor descriptions, respectively.

C.21 QUESTION 21

C.21.1 Terminology Clarification

described: includes details as to what the special processing is and why it is used.

preface block: see question 15.

C.21.2 Special Response Instructions

If there is no preface block answer D, E, or F on the basis of whether the module special processing is clearly described at any other place in the module. Answer A if there is a preface block and it is either obvious or so stated that there is no special processing which needs a description (check the source listings to verify).

If there is a description of special processing in a preface block, then keep in mind that the response anchors of A and F are only for extremely good or extremely poor descriptions, respectively.

C.22 QUESTION 22

C.22.1 Terminology Clarification

identified: listed or described in some way.

preface block: see question 15.

C.22.2 Special Response Instructions

If there is no preface block answer D, E, or F on the basis of whether module documentation information is clearly identified at any other place in the module. If there is no preface block it is highly unlikely that the documentation information will be identified anywhere else.

If there is documentation information identified within a preface block, then keep in mind that the anchor responses A and F are only for extremely complete or extremely incomplete identification, respectively. Depending upon the application and the program design, the documentation information might also include the component of which the module is a part and the reference sections for written documentation descriptions, flowcharts of the module, etc.

C.23 QUESTION 23

C.23.1 Terminology Clarification

useful: accurate, concise and lend an understanding not immediately obvious from the associated source statement(s).

C.23.2 Special Response Instructions

An answer of A or F is not likely. Keep in mind the anchor responses of A and F are for extremely good (best possible) or extremely poor (worst possible) comments.

C.24 QUESTION 24

C.24.1 Terminology Clarification

quantity: number of comments as well as the conciseness within a given comment.

legibility: readability, clarity, etc.

C.24.2 Special Response Instructions

An answer of A or F is not likely. Keep in mind the anchor responses of A and F are for the best possible quantity of comments or the worst possible quantity of comments, respectively. It is probable that "no comments" deserves an F response, at least for modules of any complexity or sheer size.

C.25 QUESTION 25

C.25.1 Terminology Clarification

explain: to make easily understood (either by comments or the natural language syntax).

C.25.2 Special Response Instructions

Answer A only if all transfers of control and destinations are clearly explained. It is not necessary that each control branch and each destination be commented. It is only necessary that it is clearly understood at each control branch under what conditions a transfer is taken and the destination of that transfer. The control structure syntax and control parameter name may in itself clearly explain a given transfer of control and its destination without further comment.

C.26 QUESTION 26

C.26.1 Terminology Clarification

machine dependencies: coding techniques which are unique to the particular computer on which the code is to execute, e.g., certain word size dependencies like FORTRAN I/O formats and shift functions.

C.26.2 Special Response Instructions

Answer A if all machine dependencies are clearly commented. It is unlikely that any assembly language module could have a reponse other than F.

C.27 QUESTION 27

C.27.1 Terminology Clarification

describe: include details as to the purpose, basic processing, etc. of the function.

uniquely recognizable function: a block of contiguous statements which perform some basic computational or logical part of the module's algorithm.

C.27.2 Special Response Instructions

The concept of module binding/strength (see question 7) is that the module should consist of a series of parts, elements, functions which have a high level of binding (hopefully functional). The understandability of each of these parts or uniquely recognizable functions is greatly aided by the inclusion of concise useful comments preceding each part. It is unlikely that this question statement should get either of the anchor responses A or F. Note that a module's preface block serves as such a descriptive comment for the overall function of the module. If there is only one uniquely recognizable module function, then by default this question statement concerns the function description within the preface block (if it exists).

C.28 QUESTION 28

C.28.1 Terminology Clarification

attributes: type, units, range, description, etc. as appropriate.

variable: variables and parameters with constant values should both be described.

describe: identify and include appropriate details.

C.28.2 Special Response Instructions

Typically all global data and module arguments are described in the preface block. Local variables are generally described in a special "declaration" section of the module. Depending upon the source language, the declaration statements may intrinsically describe some or all of a data item's attributes. Answer A only if all appropriate attributes for each variable is described in a superior manner. If none of the variables, constants, data used in this module are described, then answer F.

C.29 QUESTION 29

C.29.1 Terminology Clarification

error processing/exits: the procedure or steps taken within a module after an error condition is recognized.

identify: comment or made visibly obvious.

C.29.2 Special Response Instructions

Answer A only if all error processing/exits are clearly identified or if there are none to be identified.

C.30 QUESTION 30

C.30.1 Terminology Clarification

standard format: conventions, standards, etc. as to the physical structure of the module source listing.

module organization: placement of preface block, declarations, error exits, etc. and general format considerations such as imbedded comment format, indentations, etc.

C.30.2 Special Response Instructions

Due to the broad nature of this question statement a response of A or F is not likely.

C.31 QUESTION 31

C.31.1 Terminology Clarification

variables: variables and parameters which have constant value should be declared in a module data specification/declaration section.

declared: type (integer, real, array, etc.) is identified.

specification/declaration section: this may be a formal data declaration section or a section set up by convention for the purpose of declaring variables and other access restrictions.

C.31.2 Special Response Instructions

Answer A only if all variables used in the module are declared. The idea is that modules should only be able to use data whose attributes have been specifically identified (this includes access to global data). By restricting access, errors are avoided and by requiring data to be declared, the module descriptiveness is enhanced.

C.32 QUESTION 32

C.32.1 Terminology Clarification

None

C.32.2 Special Response Instructions

This question statement is highly subjective. However, if careful consideration has been given to the naming of global variables, distinguishing between global and local variables, and attempting to bind variables to their functional use through a descriptive name, then the time it takes to understand a module is decreased. The naming conventions which have grown out of particular applications (e.g., avionics) will also influence how functionally descriptive particular names are.

C.33 QUESTION 33

C.33.1 Terminology Clarification

logical block: control structure (see question 2)

C.33.2 Special Response Instructions

Answer A only if all control structures are indented in a manner which clearly shows the control flow. Answer A if there are no control structures other than SEQUENTIAL.

C.34 QUESTION 34

C.34.1 Terminology Clarification

statement label: address or location in the source listing which is named; ordinarily the purpose of a statement label is to provide a destination for a transfer of control.

name: the name may be numeric, alphanumeric, alphabetic, etc.

The typical method of naming statement labels to facilitate their location is by some kind of ordering. For example, FORTRAN labels are one to five digit numbers and a typical naming scheme is to keep the labels in ascending order from beginning to end of the module. Address labels in assembly language are typically alphanumeric names which can be assigned alphabetically from beginning to end of the module. Another scheme is to name labels according to a descriptive functional name which can be located according to some logical placement of the associated functions.

C.34.2 Special Response Instructions

Answer A if there are no statement labels.

C.35 QUESTION 35

C.35.1 Terminology Clarification

machine cross reference listings: the normal compiler/assembler generated cross reference information.

useful: facilitates location of variables, constants, modules calls, global versus local data references, etc.

C.35.2 Special Response Instructions

Answer F if no machine cross reference listings are available.

C.36 QUESTION 36

C.36.1 Terminology Clarification

None

C.36.2 Special Response Instructions

Answer F if there is no module flowchart or equivalent in the documentation.

C.37 QUESTION 37

C.37.1 Terminology Clarification

data flow: inputs and outputs to the module as well as any significant intermediate transformations.

C.37.2 Special Response Instructions

Answer F if there is no module flowchart or equivalent in the documentation.

C.38 QUESTION 38

C.38.1 Terminology Clarification

label: address or location which has a name; usually labels are the destination of a transfer of control. The labels in the documentation and source listings should agree in name and sequential location.

See figure IV-3 for an example of flowchart to source listing label correspondence.

C.38.2 Special Response Instructions

Answer F if there is no module flowchart or equivalent in the documentation. Answer A if there are no labels in either the documentation or the source listings. Note: general comments are not considered to be labels.

C.39 QUESTION 39

C.39.1 Terminology Clarification

inputs: global data and parameters passed in an argument list which are used within the module.

correspond: match, agree, etc.

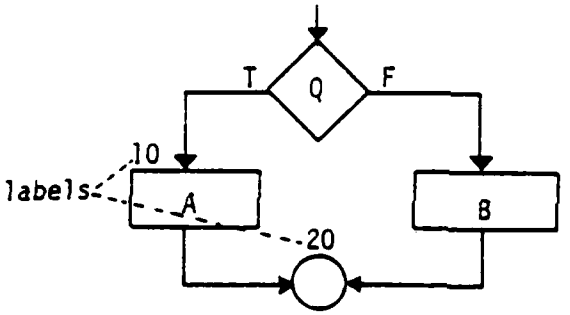
C.39.2 Special Response Instructions

Answer A only if there is a clear one-to-one correspondence between the module's inputs as described in the documentation and as shown in the source listings. If there is no description of the module's inputs in the documentation, then answer F. If the module has no inputs and this is clear from the documentation and the source listings, then answer A.

As an example of label correspondence, given the following sequence of code

```
                IF (Q) GOTO 10
                B
                GOTO 20
labels      10  CONTINUE
                A
            20  CONTINUE
```

the following flowchart segment might be generated



If, however, the equivalent sequence of code were

```
IF Q THEN A
     ELSE B;
```

then the labels on the above flowchart would be removed.

Figure IV-3. Flowchart-Source Listing Label Correspondence

C.40 QUESTION 40

C.40.1 Terminology Clarification

outputs: global data and parameters passed in an argument list which are assigned a value within the module.

correspond: match, agree, etc.

C.40.2 Special Response Instructions

Answer A only if there is a clear one-to-one correspondence between the module's outputs as described in the documentation and as shown in the source listings. If there is no description of the module's outputs in the documentation, then answer F.

C.41 QUESTION 41

C.41.1 Terminology Clarification

order of arguments: calling sequence

correspond: match, agree, etc.

C.41.2 Special Response Instructions

Answer A only if there is a clear one-to-one correspondence between the order of arguments as described in the documentation and as shown in the source listings. If there is no description of the calling sequence in the documentation, then answer F. If it is clear from both documentation and source listings that there are no arguments, then answer A.

C.42 QUESTION 42

C.42.1 Terminology Clarification

processing: control and data flow, and computations.

correspond: match, agree, etc.

C.42.2 Special Response Instructions

Answer F if the module processing is not described in the documentation.

C.43 QUESTION 43

C.43.1 Terminology Clarification

None

C.43.2 Special Response Instructions

Answer F if the documentation does not describe what conventions have been established for source code development and formatting.

C.44 QUESTION 44

C.44.1 Terminology Clarification

delineation: method of separating, highlighting, placement, etc.

The concept is that conventions should be established which more or less give a template for source code format. Adherence to the convention can be either manual or automated. This gives a uniformity to the body of the code as well as to the placement, use of, and delineation of comments which clarify the purpose of the code. It is very distracting from a consistency viewpoint to have comments delineated in a variety of ways; e.g., some have spaces, some have asterisks, some comments are imbedded, etc.

C.44.2 Special Response Instructions

Answer A if there are no comments (this is not to imply that comments are bad, but only that the delineation of the comments is indeed uniform and not distracting).

C.45 QUESTION 45

C.45.1 Terminology Clarification

data type: the set of values which a variable may assume; e.g., INTEGER, REAL, BOOLEAN, CHARACTER, ARRAY OF INTEGER, etc.

The use of a variable in bit manipulation operations (unless the variable is considered to be a packed string of bits) frequently means that the variable is being used as a type (bit string) other than its defined type.

C.45.2 Special Response Instructions

Answer A only if each variable has only one associated data type.

C.46 QUESTION 46

C.46.1 Terminology Clarification

purpose: functional use; e.g. temporary local variables are frequently used for more than one purpose under the restriction of saving storage; or the use of a variable may depend on certain control parameters, hence giving rise to multiple uses; or storage allocated in assembly and referenced by an address name (variable) may be partitioned and used in a variety of ways.

C.46.2 Special Response Instructions

Answer A only if each variable used in the module has only one functional use.

C.47 QUESTION 47

C.47.1 Terminology Clarification

global variables: data which can be accessed by more than one module.

naming convention: e.g., each global variable has a six character name, each local variable has a less-than-six character name; or, all global variables begin with the letter G, all local variables begin with the letter L.

C.47.2 Special Response Instructions

Answer F if there is no naming convention for global and local variables. Even if there is a naming convention, however, it may not be one which allows for clear distinction between global and local variables (i.e., the answer may not be A).

C.48 QUESTION 48

C.48.1 Terminology Clarification

indentation: the use of non-essential blank characters in any source line statement (usually associated with the first part of a line with executable statements).

uniform: within particular classes of statements (e.g., preface comments, imbedded comments, data declarations, control structures, etc.) there is consistency and usefulness: there should be some organizational purpose to the uniformity (namely, helps readability of the code).

C.48.2 Special Response Instructions

If there is no indentation then answer A (this is not to imply that indentation is bad, but only that the use of no indentation is indeed uniform). It should be noted that very few modules will have no indentation. An imbedded or appended comment, especially in assembly language has some indentation (or separation) from the executable statement in which it is imbedded.

C.49 QUESTION 49

C.49.1 Terminology Clarification

preface block: see question 15.

C.49.2 Special Response Instructions

If there is no preface block then answer F.

C.50 QUESTION 50

C.51.1 Terminology Clarification

high order language: other than assembly, machine, micro code, etc.; e.g. FORTRAN, PL/I, PASCAL, APL, JOVIAL,...

C.50.2 Special Response Instructions

Answer A if the source language is 100% HOL.

Answer F if the source language is 0% HOL.

Answer B-E on the basis of the following percentages:

B - $\geq$ 80%

C - $\geq$ 60%

D - $\geq$ 40%

E - $\geq$ 20%

C.51 QUESTION 51

C.51.1 Terminology Clarification

control flow: logical flow from one control structure to the next control structure (see question 2 for control structure terminology).

Logical flow which is not top to bottom is usually because of the use of unstructured control (e.g., unconditional branches and other forms of GOTO's) paths. Use of an iteration control structure is not considered to be a deviation from top to bottom control flow.

See figure IV-4 for an example and counter-example of top to bottom control flow.

C.51.2 Special Response Instructions

It can be very time consuming to scan source and clearly determine that the control flow is top to bottom (in fact, the definition may even seem ambiguous because control structures may not be easy to determine). Hence, it is unlikely to expect an answer of A.

Likewise, control does eventually get from top to bottom in most modules and usually can't be considered to be in the worst-possible-case class of examples (answer of F). Use of few statement lables is one possible indicator that the control flow might be top to bottom (at least when using a non-assembly language).

The following code segment illustrates a top to bottom control flow.

```
      DO 20, I=1, 10
      IF (Q) GOTO 10
           B
      GOTO 20
10    CONTINUE
           A
20    CONTINUE
```

The following code segment illustrates a deviation from top to bottom control flow (as well as other bad coding practices).

```
      IF (Q) GOTO 20
 5    CONTINUE
           C
      DO 10, I=1, 10
           B
      IF (R) GOTO 5
           A
10    CONTINUE
20    CONTINUE
```

Figure IV-4. Example Top to Bottom Control Flow

C.52 QUESTION 52

C.52.1 Terminology Clarification

None

C.52.2 Special Response Instructions

Answer A only if there is no extraneous code.

C.53 QUESTION 53

C.53.1 Terminology Clarification

None

C.53.2 Special Response Instructions

Answer A only if there are no specialized coding techniques used in the module.

C.54 QUESTION 54

C.54.1 Terminology Clarification

See figure IV-5.

C.54.2 Special Response Instructions

It is unlikely that the evaluator would be able to determine except through a detailed study of the source listings whether clever programming techniques were completely avoided or whether the module deserves to rank with the worst in this respect. Hence, responses at the anchor points A or F are fairly unlikely.

The following example of clever FORTRAN programming is illustrative of an unfortunately large set of possible examples.

```
      DO 10 I=1, N
      DO 10 J=1, N
      A(I,J) = (I/J)*(J/I)
10    CONTINUE
```

In this example a real matrix is being initialized. However, when I>J then J/I = 0 due to intrinsic FORTRAN integer division rules and likewise, when J>I then I/J = 0. The net result is that all diagonal elements, A(I,I), of the matrix are assigned the value 1.0 and all other elements are assigned the value 0.0. Clever, but a more understandable and more efficient version is given below.

```
C     INITIALIZE -A- TO BE THE IDENTITY MATRIX
      DO 20 I=1, N
      DO 10 J=1, N
      A(I,J) = 0.0
10    CONTINUE
      A(I,I) = 1.0
20    CONTINUE
```

Figure IV-5. Example of Clever Programming

C.55 QUESTION 55

C.55.1 Terminology Clarification

GOTO-like branch statements: control structures with conditional or unconditional branching to a source label.

essential: part of the basic structured control constructs (see question 2) or providing an ESCAPE or BREAK capability; the ESCAPE is simply a branch from within a given control construct to the statement immediately following the control construct (or nest of control constructs).

See figure IV-6 for examples of essential uses of GOTO branches.

C.55.2 Special Response Instructions

None

C.56 QUESTION 56

C.56.1 Terminology Clarification

statement label: address or location to which control can be transferred within a module.

use: number of statement labels as well as the number of times the labels serve as the destination of a transfer of control.

C.56.2 Special Response Instructions

If there are no statement lables, then answer A.

C.57 QUESTION 57

C.57.1 Terminology Clarification

basic algebra: functions, equations, polynomials, graphing of functions, basic manipulations, etc.; excludes calculus, differential equations, Fourier transforms, statistical algorithms, etc.

C.57.2 Special Response Instructions

None

The use of GOTO to construct some of the essential control constructs is illustrated in question 2 guidelines. The use of a computed GOTO to construct a CASE construct and the use of a GOTO as an escape are illustrated below. Similar constructs can be generated in assembly language.

CASE:

```
      IF((M.LT.1) .OR. (M.GT.4)) GOTO 50
      GOTO (10,20,30,40), M
10    CONTINUE
      A
      GOTO 50
20    CONTINUE
      B
      GOTO 50
30    CONTINUE
      C
      GOTO 50
40    CONTINUE
      D
50    CONTINUE
```

ESCAPE:

```
      DO 20 I=1, 1000
        DO 10 J=1, 1000
        A
        IF (Q) GOTO 30
10      CONTINUE
20    CONTINUE
30    CONTINUE
```

Figure IV-6. Example of Essential Use of GOTO

C.58 QUESTION 58

C.58.1 Terminology Clarification

number: count, amount of use, and level of nesting.

Data structures are either primitive data types (e.g., INTEGER, REAL, BOOLEAN, CHAR, etc.) or are structures with components. The component type may be primitive or may itself be a structure. The only non-nested data structures are the primitive data types. Examples of nested structures would be arrays, arrays of arrays, records, FORTRAN COMMON (with more than one primitive component), etc. In assembly language the use of a block of storage as an array (e.g., via manual index computations) would constitute a nested data structure. However, for the most part, assembly language modules will show a higher response (closer to A) on this question and lower responses on question 1 and 64. Note that the use of fields (e.g., bit strings) within a computer word is considered to be a nested data structure.

C.58.2 Special Response Instructions

Consider A only if there is no use of nested data structures. The concept is that reasonable use of a small number of primitive structures is simple and that deviations (whether because of necessity or other reasons) such as the use of the nested structures tends to reflect more complexity. Of course, this concept can be carried to extreme. If the software developer has done a reasonable design, then the type of data structures used should reflect the functional use of the data and hence the complexity of the programming task. This is not always the case. It should be noted, however, that if there are an excessive number of primitive variables (when in fact the related variables should have been grouped into structures), then the questions 1 and 64 should score poorly. This balances the simplicity issue and reflects properly on the poor modularity of the data.

C.59 QUESTION 59

C.59.1 Terminology Clarification

number: count and level of nesting.

control structures: see question 2.

compound: contains one or more of the primitive control structures (see question 2).

See figure IV-7 for examples of compound control structures.

C.59.2 Special Response Instructions

Answer A only if there are no nested control structures. An answer of F should reflect an extremely poor control structure organization. It is not possible to specify precise guidelines for the anchor F response. However, the greater number of nested control structures and/or the greater the level of nesting for any individual nest should be reflected through a lower response (A highest, F lowest).

C.60 QUESTION 60

C.60.1 Terminology Clarification

executable statement: as defined by the language; for our purposes this excludes comments, data declarations, and variable/constant declarations.

C.60.2 Special Response Instructions

The response should be based on a relative average of the number of source lines which contain only one executable statement to the total number of source lines with at least one executable statement.

A - 0% no source line has more than one executable statement.

B - $\geq$ 20%

C - $\geq$ 40%

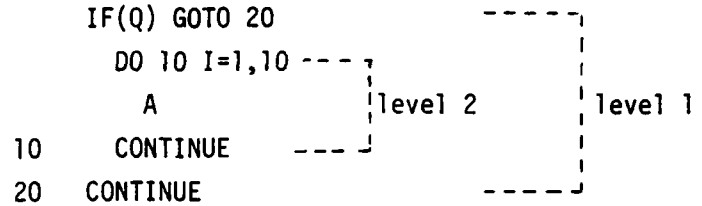
D - $\geq$ 60%

E - $\geq$ 80%

F - 100% each source line has more than one executable statement.

Examples of compound control structures are given below. Each example illustrates one compound structure with a level number.

EXAMPLE 1:



EXAMPLE 2:

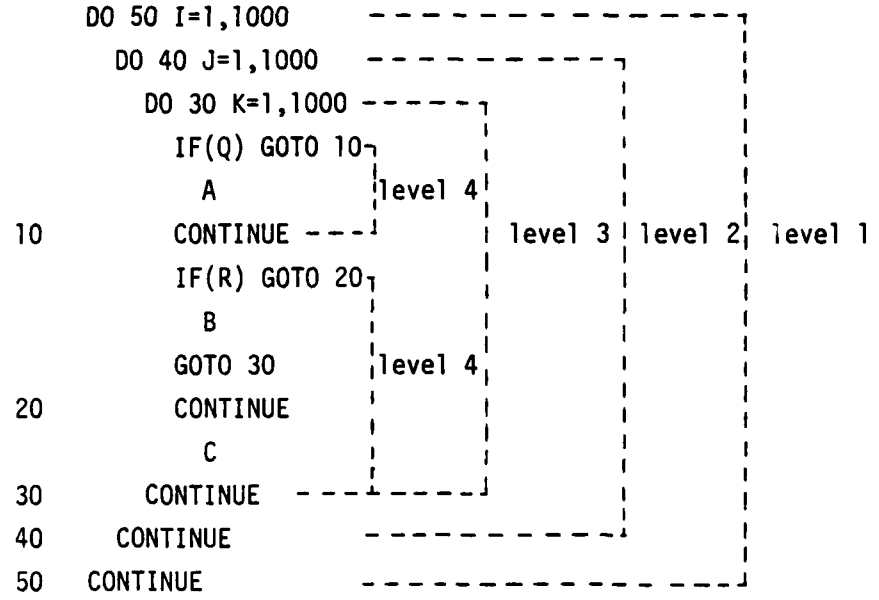


Figure IV-7. Examples of Compound Control Structures

C.61 QUESTION 61

C.61.1 Terminology Clarification

See figure IV-8.

C.61.2 Special Response Instructions

Answer A only if there are no compound Boolean expressions.

C.62 QUESTION 62

C.62.1 Terminology Clarification

number: count

(control) expression: IF, CASE, or other decision control expression. DO, DO-WHILE, or other iterative control expression.

See figure IV-9 for examples of counting (control) expressions.

C.62.2 Special Response Instructions

The count of control expressions is closely related to the number of independent control cycles in a module. The more control expressions there are the more complex the control logic tends to be. The following guidelines will anchor the A and F responses, but are fairly subjective (especially the F anchor). The guidelines for the A response is suggested from other independent research. Remember to count all repetitions of the same control expression also.

Answer A if count $\leq$ 10.

Answer F if count > 50.

Compound Boolean expressions are primarily used in high order languages where Boolean operators such as AND and OR are available. Some examples of compound Boolean expressions are illustrated below.

```
A.AND.B
(A.AND.B).OR.C
(A.AND.B).AND.C
```

A typical use might be as follows:

```
IF((I.GT.1).AND.(J.LT.10)) GOTO 20
```

Figure IV-8. Examples of Compound Boolean Expressions

The following examples indicate how to count the control expressions.

CONTROL STRUCTURE	STATEMENT	CONTROL EXPRESSION	COUNT
Decision	IF (A.OR.B) GOTO 10	A;B	2
	IF (A.AND.B) GOTO 10	A;B	2
	IF (C.GT.D) GOTO 10	C.GT.D	1
	IF ((A.AND.B).OR.(C.GT.D)) GOTO 10	A;B;C.GT.D	3
	CASE (I) OF	I=1;I=2;I=3	2
	1: A	(alternatives)	(number of
	2: B		alternatives
	3: C		less one)
	END CASE		
Iteration	DO 10 I=1,10	I.LT.1	2
	A	I.GT.10	
10	CONTINUE		
	DO	Q	1
	A		
	WHILE Q		
	DO	A;B	2
	A		
	UNTIL (A.OR.B)		

Figure IV-9. Examples of Counting Control Variables

C.63 QUESTION 63

C.63.1 Terminology Clarification

See table IV-2 for guidelines for determining operators.

See figure IV-10 for an example of counting operators and operands in FORTRAN and in PDP assembly code.

C.63.2 Special Response Instructions

The concept is that the more operators there are, the more discriminations one must make in order to understand the module's function. It is not intended that the evaluator spend a great amount of time counting operators. In a period of a few minutes using the above guidelines it should be possible to obtain a reasonable estimate of the number of unique operators. The anchors below for A and F responses are not sacred, but are reasonable.

Answer A if count $\leq$ 10.

Answer F if count > 50.

Keep in mind it is the total number of unique operators which are counted, not the total number of operators (which includes the repetitious use of each unique operator).

C.64 QUESTION 64

C.64.1 Terminology Clarification

operands: variables and constants

C.64.2 Special Response Instructions

The concept is that the more operands there are, the more discriminations one must make in order to understand the module's function. It is not intended that the evaluator spend a great amount of time counting operands. In a period of a few minutes it should be possible to obtain a reasonable estimate of the number of unique operands. The anchors below for A and F responses are not sacred, but are reasonable.

Answer A if count $\leq$ 40.

Answer F if count > 240.

TABLE IV-2. Guidelines for Determining Operators

The minimal number of operators for any algorithm is 2 (one operator is the name of the algorithm by which it is invoked and an assignment or grouping symbol for transfer of the result, e.g. Y = RANDOM or CALL RANDOM (Y)).

The following are guidelines as to what constitutes an operator. Some examples are also provided. The list below should be considered representative, not complete.

1. All typical language verbs are operators;
   e.g. unary op: negation (-), set complement (')
   binary op: addition (+)
   subtraction (-)
   multiplication (*)
   division (/)
   exponentiation (**)
   assignment (=)
   relation op: less than (LT, <)
   greater than (GT, >)
   equal (EQ, =)
   not equal (NE, ≠)
   less than or equal (LE, <=)
   greater than or equal (GE, >=)
   logical op: and
   or
   not
   control op: decision (IFTHENELSE, FORTRAN logical/arithmetic, assembly, branch, CASE)
   iterative (DO loop, DOWHILE, DOUNTIL, etc.)
   sequential grouping (BEGINEND, DO)
   go to (each district go to label is a unique operator)
2. There are various groupings of terms, each of which is an operator.
   grouping: expression
   subscript
   argument list
3. Each reference call is an operator.
   external: module/function call (each unique one is counted)
   intrinsic function call (MOD,MAX, ABS, SHIFT, etc.)
   external function call (SIN, COSINE, LOG, EXP, SQR, etc.)
4. Various delimeters are considered to be operators.
   delimeters: subscript comma
   argument list comma
   do loop comma
   end of statement (end of card, semicolon, etc.)

DISPLAY I

```
      SUBROUTINE SORT (X, N)
      DIMENSION X(N)
      IF (N .LT. 2) RETURN
      DO 20  I = 2,N
         DO 10 J = 1,I
            IF (X(I) .GE. X(J)) GO TO 10
            SAVE = X(I)
            X(I) = X(J)
            X(J) = SAVE
   10 CONTINUE
   20 CONTINUE
      RETURN
      END
```

TABLE 1. Operators of the interchange sort program of Display I.

	Operator	*Count*
1	End of statement	7
2	Array subscript	6
3	=	5
4	IF ( )	2
5	DO	2
6	,	2
7	End of program	1
8	.LT.	1
9	.GE.	1
$n_1 = 10$	GO TO 10	1
		$28 = N_1$

TABLE 2. Operands of the interchange sort program of Display I.

	Operand	*Count*
1	X	6
2	I	5
3	J	4
4	N	2
5	2	2
6	SAVE	2
$n_2 = 7$	1	1
		$22 = N_2$

DISPLAY II

```
                .TITLE    SORT
SORT::
                MOV       (R5)+,R0        ;CHECK # OF ARGUMENTS.
                CMP       #2,R0
                BEQ       10$             ;TWO IF HERE.
                JMP       70$             ;BAD # OF ARGUMENTS HERE.
10$:            MOV       (R5)+,X         ;SAVE ADDRESS OF "X" ARRAY.
                MOV       @(R5)+,R1       ;MOVE CONTENTS OF "N" INTO R1.
                CMP       #2,R1           ;IS (N .LT.2)?
                BGE       70$             ;YES.
                MOV       #2,R2           ;SET UP "I".
20$:            CMP       R2,R1           ;IS (I .GT.N)?
                BMI       70$             ;YES. WE'RE THROUGH.
30$:            MOV       #1,R3           ;SET UP "J".
40$:            CMP       R3,R2           ;IS (I .GT. J)?
                BMI       60$             ;YES.
                CMP       X(R2),X(R3)     ;IS (X(I) .GE. X(J))?
                BPL       50$             ;YES.
                MOV       X(R2),SAVE      ;NO. SWITCH VALUES.
                MOV       X(R3),X(R2)
                MOV       SAVE,X(R3)
50$:            INC       R3
                BR        40$
60$:            INC       R2
                BR        20$
70$:            RTS       PC              ;RETURN TO CALLER.
X:              .WORD     0               ;DATA STORAGE -- ADDRESS OF "X".
SAVE:           .WORD     0               ;DATA STORAGE.
                .END
```

TABLE 3. Operators of the interchange program in assembly language.

	Operator	*Count*
1	End of statement	17
2	,	9
3	Apply index register	6
4	MOV	5
5	CMP	4
6	INC	2
7	End of program	1
8	BMI 70$	1
9	BMI 60$	1
10	BPL 50$	1
11	BR 40$	1
12	BR 20$	1
$n_1 = 13$	BGE 70$	1
		$50 = N_1$

TABLE 4. Operands of the interchange program in assembly language.

	Operand	*Count*
1	R2	7
2	R3	6
3	X	5
4	#2	2
5	R1	2
6	SAVE	2
$n_2 = 7$	#1	1
		$26 = N_2$

Figure IV-10. Examples of Counting Operators and Operands

C.65 QUESTION 65

C.65.1 Terminology Clarification

executable statements: as defined by the language; for our purposes this excludes comments, data declarations, and variable/constant declarations.

C.65.2 Special Response Instructions

The following guidelines for anchor responses are not sacred, but are reasonable.

Answer A if count $\leq$ 50.

Answer F if count > 300.

C.66 QUESTION 66

C.66.1 Terminology Clarification

functions: submodules, groups of related statements, etc. as appropriate.

I/O functions: actual external device interfaces; e.g., FORTRAN read and write, operating system file management calls, hardware contoller interface calls, etc.

application functions: any functions which provide specific operational computations.

C.66.2 Special Response Instructions

The concept is that mixing of I/O code and other operational computations makes it difficult to modify either the I/O or the operational computations which probably are bound to the I/O in some manner. By keeping the mix to a minimum it is easier to maintain. Answer A only if there is no mix (i.e., the functions are either entirely I/O or entirely non-I/O).

C.67 QUESTION 67

C.67.1 Terminology Clarification

functions: submodules, groups of related statements, etc. as appropriate.

machine dependent functions: functions which are particular to the host computer; e.g. architectural assumptions.

application functions: any functions which provide specific operational computations.

C.67.2 Special Response Instructions

The concept is that mixing of machine dependent code and with other operational computations makes it difficult to modify either the machine dependent code or the operational computations. Answer A only if there is no mix (i.e., the functions of this module are either entirely machine-dependent or non-machine-dependent). Answer F only if the module's source language is 100% assembly.

C.68 QUESTION 68

C.68.1 Terminology Clarification

repeatedly: at least three times.

parameterized: referenced by address or name, not by the actual constant value; e.g., PI instead of 3.1415926.

C.68.2 Special Response Instructions

Answer A only if all constants used at least three times have been parameterized.

C.69 QUESTION 69

C.69.1 Terminology Clarification

self-modifying code: code which changes the actual code instructions of some other part of the module or of some other module depending upon the current processing state; dynamic code generation.

relative addressing: used most often when source language is assembly.

Another example of processing dependence is a module whose effect is dependent upon the number of times it has been executed. Frequently a module performs only certain processing functions the first time or the Nth time it is executed.

C.69.2 Special Response Instructions

Answer A only if there is no use of code whose effect is contingent upon some processing dependency. The concept is that code should not have built-in processing dependencies which make code modification difficult. These processing dependencies may be internal module dependencies, may be code which generates dependencies in other modules, or may be code which depends upon some external processing state for its functional effect.

C.70 QUESTION 70

C.70.1 Terminology Clarification

parameterized: referenced by address or name, not by actual constant value.

Some examples would be the use of an array size to control the number of iterations, or to control indexing into the array or other storage locations. Since it may be necessary to modify the size of a given data structure, it is very helpful not to have to be concerned with catching all the implicit references to the structure size within the algorithm code which uses the data structure.

C.70.2 Special Response Instructions

Answer A only if there is no use of data structure size information (implicit or explicit) within the processing logic of the module except where the size has been parameterized.

C.71 QUESTION 71

C.71.1 Terminology Clarification

parameterized: referenced by address or name, not by actual constant value.

The concept is that any metric attribute (or specific requirement) which in some manner controls the processing logic of the module should not be used as a literal constant, but should be parameterized to allow for modification ease should the necessity arise.

C.71.2 Special Response Instructions

Answer A only if all attributes which affect processing are parameterized either locally or globally.

C.72 QUESTION 72

C.72.1 Terminology Clarification

timing allocation: timing requirement/specification.

C.72.2 Special Response Instructions

Answer A only if the module is non-real-time or if there is at least 25% timing margin.

C.73 QUESTION 73

C.73.1 Terminology Clarification

The concept is that it would be best if a module does not have to be modified simply because more data is to be processed. For example, a sort algorithm may well be volume bound or may not be depending upon how the processing algorithm is set up to handle the input and output of the data.

C.73.2 Special Response Instructions

None

C.74 QUESTION 74

C.74.1 Terminology Clarification

functional part: submodule, task, contiguous statements performing a basic computational step, or even the complete module.

The evaluator should consider the level of functional part to be inserted, deleted or replaced as applicable on the basis of the individual module's purpose and component functions. The idea is that if it is at all possible that the module or its functional parts might be required to be modified on a functional insert/delete/replace basis how accommodating is this module. As an example, a trigonometric function is usually quite replaceable.

C.74.2 Special Response Instructions

None

C.75 QUESTION 75

C.75.1 Terminology Clarification

array: either an explicitly declared array or a storage area (e.g., in assembly) which is effectively used as an array.

subscript: index.

C.75.2 Special Response Instructions

If the module does not use arrays or some other type of indexing which could exceed the actual maximum size, then answer A.

C.76 QUESTION 76

C.76.1 Terminology Clarification

None

C.76.2 Special Response Instructions

If the module does not contain any operations which would give an undefined or garbage result, then answer A.

C.77 QUESTION 77

C.77.1 Terminology Clarification

None

C.77.2 Special Response Instructions

Answer A only if there is no code in the module which would require detailed testing (because of sheer logic or theoretical concepts).

C.78 QUESTION 78

C.78.1 Terminology Clarification

None

C.78.2 Special Response Instructions

None

C.79 QUESTION 79

C.79.1 Terminology Clarification

The basic question is whether all input data is checked as appropriate for format and range (or any other attribute which might make the data invalid). It is especially important to validate input data which is a logic control parameter (e.g., decision parameter, loop index) or could cause an undefined operation (e.g., divide by zero). If the validation is properly accomplished, then are appropriate diagnostic messages and/or error codes recorded or reported.

C.79.2 Special Response Instructions

It may be that all data which is input to this module either does not require validation or is validated in some other (perhaps one module for that purpose) place; if it is clear from the comments in the source listings that this is the case, then answer A.

C.80 QUESTION 80

C.80.1 Terminology Clarification

internal module failure: algorithm failure due to error conditions which are recognized within the module; these error conditions may be due to out-of-bound array subscript, undefined operations, algorithm inadequacy, etc. This failure is not due to error conditions from input data checks, but may be based upon perfectly valid input data.

C.80.2 Special Response Instructions

None

C.81 QUESTION 81

C.81.1 Terminology Clarification

intermediate results: input data is transformed to output data through a series of intermediate steps; at each step data, or intermediate results, may be useful in determining where processing correctness begins and ends.

display: print, hard copy, video, etc.

selectively: can be displayed or not displayed by user-specified options (dynamic or static).

C.81.2 Special Response Instructions

If a module has no significant intermediate results (perhaps because of its inherent simplicity) which would be useful to know in order to test/retest the module, then answer A.

C.82 QUESTION 82

C.82.1 Terminology Clarification

aids: test probes, performance probes, etc. which may be in the form of language processor generated code, program diagnostic modules designed specifically for that purpose, or actual inline code which can be selectively activated by the user; it may be that event trace performance information is automatically collected and monitored by a separate program task; language features such as conditional compilation can greatly influence the capability for a design to include good instrumentation aids for the purpose of collecting trace information or intermediate data results (see question 81).

C.82.2 Special Response Instructions

None

C.83 QUESTIONS 83-89

C.83.1 Terminology Clarification

See evaluator guideline handbook.

C.83.2 Special Response Instructions

The evaluator should not average any previous responses to obtain a response to the general questions.

For each specific maintainability test factor, the module source listing questionnaire has addressed some associated module source listing characteristics. There may be more important characteristics for the particular application, or the manner in which the included questions were asked may not have allowed the evaluator to express a most appropriate answer.

The general questions should be answered with respect to what the evaluator's impression was concerning the extent to which each specific test factor contributes to maintainability.

And, independent of how maintainability has been partitioned into test factors, the evaluator should give a response as to the extent to which the module's source listing characteristics will contribute to overall maintainability of the module.

D. EXAMPLES

There are two extended examples which are included in this section. The first example (figure IV-11) consists of an outline for a software product specification based on MIL-STD-483 and MIL-STD-490. The second example consists of a module source listing (figure IV-12), flowchart (figure IV-13), and a table of suggested responses to the source listing questionnaire (table IV-3) applied to this module. The module is not meant to be an example of "best" coding practices. For illustrative purposes there are some parts which have not been or cannot be made to receive a "best" response. And, keep in mind, the responses are subjective.

SOFTWARE PRODUCT SPECIFICATION

This Appendix contains a composite of MIL-STD-483 and MIL-STD-490 which provide a format for the Software Product Specification. The term CPCI stands for computer program configuration item.

1.0 SCOPE

1.1 Identification. This paragraph shall contain the approved indentification, nomenclature and authorized abbreviation for the computer program. This section of the specification shall begin with the following opening phrase: "This part of this specification establishes the requirements for performance, design, test, and qualification of a computer program identified as (insert nomenclature and configuration item number). This computer program is used to (provide) (accomplish). . . ."

1.2 Functional Summary. This paragraph shall contain a summary of the purpose of the specification and a brief description of the overall computer program by major functions (tasks). It shall further identify and summarize the specification content, composition, and intent.

2.0 APPLICABLE DOCUMENTS

All and only those documents referenced in Sections 3, 4, and 5 and Appendixes of the specification shall be listed in Section 2 of the specification. If numerous, Section 2 may reference an appendix or other appropriate documents containing a complete listing. References shall be confined to documents currently available at the time of issuance of the current revision of the specification. Figures bound integrally with the specification shall not be listed in Section 2.

2.1 Government Documents. The following documents of the exact issue shown form a part of this specification to the extent specified herein. In

Figure IV-11. Software Product Specification Outline

the event of conflict between the documents referenced herein and the contents of this specification shall be considered a superseding requirement.

Government documents shall be listed in the following order:

SPECIFICATIONS:

Federal
Military
Other Government Activity

STANDARDS:

Federal
Military
Other Government Activity

OTHER PUBLICATIONS:

Manuals
Regulations
Handbooks
Bulletins

2.2 Non-Government Documents. Begin with paragraph 2.1 above. Non-Government documents shall be listed in the following order:

SPECIFICATIONS

STANDARDS

OTHER PUBLICATIONS

Figure IV-11. Software Product Specification Outline (Continued)

3.0 REQUIREMENTS

This section shall contain performance and design requirements for the CPCI. It shall further include the functional requirements for the CPCI and establish those requirements which normally will be verified during category I, or equivalent, test. This section shall also define the CPCI and specify design constraints and standards necessary to assure compatibility of the CPCI with other computer programs and equipments. Performance and design requirements to be included herein shall be allocated from, identical with, or in recognition of, requirements established by the system/system segment specification.

Requirements included in the system/system segment specification, which are directly related to requirements specified herein, may be incorporated by reference. Requirements shall be specified to the level of detail necessary to establish limits for design. Quantitative requirements shall be within the three principal subparagraphs included herein. The introductory paragraph shall include a general description of the CPCI and its function within the system/equipment to which it applies.

3.1 Computer Program Definition. This paragraph shall, in subparagraphs included herein, specify the functional relationship of the CPCI to other equipment/computer programs and identify Government-furnished computer programs incorporated in the CPCI. General and/or descriptive material may be included in basic-paragraph 3.1.1. Quantitative requirements shall be included in the subparagraphs included herein.

NOTE: Interfaces defined in this section shall include, at a minimum, all relevant characteristics of the computer, such as memory size, word size, access and operation times, interrupt capabilities, and special hardware capabilities. The computer characteristics may be described by references to the applicable documentation and descriptions. If the compiler/assembler is a Government-furnished component to be incorporated into this CPCI, it

Figure IV-11. Software Product Specification Outline (Continued)

shall be referenced in subparagrapn 3.2.3.2. If the compiler/assembler is to be constructed as part of the development of the CPCI, the language characteristics shall be defined under paragraph 3.2, Detailed Functional Requirements.

3.1.1.1 Interface Block Diagram. The relationship of the CPCI to other equipment/computer programs with which it must interface shall be graphically portrayed in this paragraph. This paragraph shall incorporate, in subparagraphs as appropriate, a functional block diagram or equivalent representation of the interface requirements of the CPCI. The graphic portrayal of the CPCI shall be accomplished to the level of detail necessary to identify the functional interfaces between the CPCI and other identified equipment/computer programs.

3.1.1.2 Detailed Interface Definition. This paragraph shall specify, in subparagraphs as appropriate, the functional relationship of the CPCI to interfacing equipment and computer programs. This information shall be given in qunatitative terms with tolerances where applicable to the level of detail necessary to permit design of the CPCI. Functional interfaces shall specify the input/output requirements of the CPCI in terms of data rate, message format, etc. In addition, this paragraph shall specify design requirements imposed upon other equipment/computer programs as a result of the design of the CPCI (e.g., operator console equipment, display characters, junction and distribution boxes, terminal boards, etc.).

3.2 Detailed Functional Requirements. This paragraph shall specify, in subparagraphs defined below, the functional requirements of the CPCI. Requirements shall be stated in quantitative terms, with tolerances where applicable. General and descriptive material may be included in basic paragrapn 3.2, which shall incorporate either directly or by reference, a functional block diagram or equivalent representation of the CPCI. The graphic portrayal shall be accomplished to the level of detail necessary to

Figure IV-11. Software Product Specification Outline (Continued)

illustrate the functional operation of the CPCI, the relationships between these functions, and the relationships between the functions and other identified system/equipment functions. This diagram is not intended to be restrictive on computer program detail design. Requirements for separately identified CPCI functions shall be described in subsequent paragraphs as appropriate. A subparagraph shall be included for each operational function, plus special functions such as sequencing control, displays, error detection and recovery, input and output control, real time diagnostics, operational data recording, etc. The descriptions of these CPCI functional requirements shall include ther relative sequencing, periodicities, options, and other important relationships of each as appropriate. Paragraphs 3.2.X and subparagraphs shall be repeated for each function above.

3.2.X Function X. The basic paragraph for each function shall begin with descriptive and introductory material which defines the function and its relationship to other functions. Then, the following three subparagraphs shall specify the quantitative requirements concerning the function.

3.2.X.1 Inputs. This paragraph shall specify either directly or by reference to another part of this specification the source(s) and type(s) of input information associated with a function of the CPCI. This shall include a description of the information, its source(s) and, in quantitative terms, units of measure, limits and/or ranges of units of measures, accuracy/precision requirements, and frequency of input information arrival.

3.2.X.2 Processing. This paragraph shall provide a textual and mathematical description of each of the processing requirements of each function. Presentation of the mathematical descriptions under each function shall include:

a. Purpose -- This area shall describe the exact intent of the mathematical operation(s). This involves a definition of the specific input and output parameters and the processing required.

Figure IV-11. Software Product Specification Outline (Continued)

b. Approach -- This area shall contain a textual description of each mathematical operation specified. The accompanying narrative shall identify accuracies required, sequence and timing of events, and relevant restrictions or limitations. Derived equations shall be shown with appropriate mathematical and control symbols adequately defined.

3.2.X.3 Outputs. This paragraph shall specify, either directly or by reference to another part of this specification the destination(s) or type(s) of output information associated with a function of the CPCI as a result of the processing described in paragraph 3.2.X.2. This shall include a description of the information; its destination(s); and, in quantitative terms, units of measure, accuracy/precision requirements, frequency of output information, etc., where applicable.

3.2.n Special Requirements. This paragraph shall specify, in appropriate subparagraphs, requirements which affect the design of the CPCI and are distinguishable from the performance requirements of paragraph 3. These requirements result from general considerations of CPCI usability. These may include, but are not limited to, requirements for:

a. The use of programming standards to assure compatibility among computer program components (CPCs - subprogram or groups of subprograms).

b. Program organization, such as overall program segmentation. In addition, for CPCIs which contain or process classified information, special attention shall be given to the requirements for protecting classified information.

c. Program design resulting from consideration of modifications to the CPCI during operation (e.g., on-site modification requirements and the permissible amount of operational degradation allowed during installation of modification may be specified).

Figure IV-11. Software Product Specification Outline (Continued)

d. Special features, to facilitate the testing of the CPCI. For example, special procedures for the design of the CPC interfaces, requirements for intermediate printouts, and commentary on the program listing may be required.

e. Expandability (growth potential) to facilitate modifications and additions to the CPCI.

3.2.n.1 Human Performance. Human performance/human engineering requirements for the CPCI shall be specified in this paragraph (e.g., minimum times for human decision-making, maximum time for program responses, maximum display densities of information, clarity requirements for displays, etc.). For CPCIs which directly support a system(s), this paragraph shall cite the appropriate paragraph(s) of the system/system segment specification which establish the human performance/human engineering requirements for all system equipment, and incorporate requirements peculiar to this CPCI on an add and/or delete basis.

3.2.n.2 Government-Furnished Property List. This paragraph shall list the Government-furnished computer programs which the CPCI must be designed to incorporate. The list shall identify the program by nomenclature; specification number; model number, if appropriate; and associated documentation.

3.3 Adaptation. These paragraphs shall specify, in descriptive and quantitative terms, the data base requirements which affect the design of the CPCI. In addition, where applicable, these paragraphs shall specify the methods necessary to convert these parameters into a form suitable for use by the computer program. These requirements are divided into three classes: general environment, system/equipment parameters, and system/equipment capacities, and shall be presented as follows:

3.3.1 General Environment. This paragraph shall contain a description of environmental data detailing the characteristics anticipated for all

Figure IV-11. Software Product Specification Outline (Continued)

particular installations. Each installation will select and set the required data and value for operational use. Examples of such data are: grid limits, radar ranges and areas of coverage, prescribed safety limits, etc.

3.3.2 System Parameters. This paragraph shall contain a description of constants required by one or more subprograms that may change from time to time incrementally within a specified range according to operational needs. Such data consists of allowable trajectory deviations, missile performance characteristics, ranges of possible values, accuracy/precision and quantities, etc.

3.3.3 System Capacities. This paragraph shall contain a description of the capacity requirements for the computer program. Items such as compatibility for total simultaneous missile trajectory controls, total number of simultaneous displays and operator station requests, track capacities number and types of inputs processed, etc., shall be described. The system capacities are directly related to computer storage capacities, interfacing subsystem timing rates, and interfacing equipment capacities.

4.0 QUALITY ASSURANCE PROVISIONS

Requirements for formal verification of the performance of the CPCI in accordance with the requirements of Section 3 of this specification shall be specified in this paragraph. Formal verification of performance of the CPCI shall determine acceptance of the CPCI. This paragraph shall specify formal verification requirements to a level of detail which:

a. Designates verification requirements and methods in Section 4 for performance and design requirements in Section 3. The methods of verification to be specified herein may include inspection of the CPCI, review of analytical data, demonstration tests, and review of test data.

Figure IV-11. Software Product Specification Outline (Continued)

b. Specifies requirements for verification to the level of detail necessary to clearly establish the scope and accuracy of the test method.

c. Permits ready identification of each verification requirement specfied in Section 4 with the appropriate performance, design requirement paragraph in Section 3.

d. Allocates verification requirements to the subparagraphs included herein.

NOTE: This section shall not incorporate, either directly or by reference, detail test planning documentation and operating instructions. Requirements specified herein shall be the basis for preparation and validation of such documents.

4.1 Introduction. This paragraph shall establish the requirements which provide the basis for development of a test plan and test procedures for the subject program. All test/verification requirements shall be specified within the subparagraphs included herein.

4.1.1 Category I Test. The term "Category I Test" as used herein is defined to include all testing of the CPCI other than that accomplished during the formal Catgeory II (or equivalent) system/configuration item test programs. (See paragraph 4.1.5 below.) Category I testing is subdivided into the following broad types:

a. Computer program test and evaluation. Tests conducted prior to and in parallel with preliminary or formal qualification tests. These tests are oriented primarily to support the design and development process.

b. Preliminary qualification tests. Formal tests oriented primarily towards verifying portions of the CPCI prior to integrated testing/formal

Figure IV-11. Software Product Specification Outline (Continued)

qualification tests of the complete CPCI (see paragraph 4.1.3 below). These tests will typically be conducted at the contractor's design and development facilities.

c. Formal qualification tests. Formal tests oriented primarily towards testing of the integrated CPCI, normally using operationally configured equipment at the category II site prior to the beginning of category II testing. This testing will emphasize those aspects of the CPCI performance which were not verified by preliminary tests. The testing requirements which cannot be verified during category I test shall be specified in paragraph 4.1.5.

NOTE: Requirements for verification included in the system/system segment specification, which are directly related to requirements specified herein, may be incorporated herein by reference to avoid redundant establishment of the requirements.

4.1.2 Computer Programming Test and Evaluation. This paragraph shall contain the following:

a. Programming test and evaluation which satisfy one or both of the criteria listed below shall be included herein. (Routine tests accomplished in support of design and development, which do not satisfy one or both of these criteria, shall not be specified herein.)

(1) They are intended to be the only source of data to qualify specific requirements in Section 3.

(2) They must be accomplished as part of an integrated test program involving other systems/equipment/computer programs (e.g., verification of requirements in paragraph 3.1.1).

Figure IV-11. Software Product Specification Outline (Continued)

4.1.3 Preliminary Qualification Tests. This paragraph shall specify only those preliminary qualification test requirements which require formal recognition by the Air Force and are oriented toward verifying proper performance of portions of the CPCI prior to integrated testing of the complete CPCI. Testing accomplished by the contractor in support of design and development which does not require recognition by the Air Force, other than it is within the general terms and conditions of a contract, shall not be specified herein. Requirements for preliminary qualifications specified herein shall reference requirements in Section 3.

4.1.4 Formal Qualification Tests. This paragraph shall specify requirements for formal qualification tests of the integrated CPCI to demonstrate and/or verify that the requirements established in Section 3 have been satisfied. This paragraph shall, in subparagraphs as appropriate, specify the requirements and method of verification for the requirements specified in Section 3, with the following exceptions:

a. The requirement in Section 3 has been identified, and verification that is has been satisfied by one of the tests included in paragraphs 4.1.2 and 4.1.3.

b. The requirement in Section 3 is peculiar to category II type system testing and will be identified in paragraph 4.1.5.

Verification of the requirements may be accomplished by inspection, demonstration, test, and review of test data, or combinations of these. This paragraph shall contain a subparagraph for each of the principal methods of verification, and shall specify therein the requirements of Section 3 to be verified by the method.

Figure IV-11. Software Product Specification Outline (Continued)

4.1.5 Category II System Test Program. This paragraph shall identify requirements specified in Section 3 which cannot be verified until Category II testing (or equivalent) and must be listed as a Category II test requirement.

4.2 Test Requirements. This paragraph shall specify the requirements for each type of testing. The requirements shall include test formulas, algorithms, techniques and acceptable tolerance limits, as applicable.

5.0 PREPARATION FOR DELIVERY

This section is normally not applicable.

6.0 NOTES

This section shall include information which is stated here for administrative convenience only, and is not a part of the specification for the CPCI in the contractual sense (i.e., it shall not include requirements which constrain design, development, and qualification of the CPCI and require compliance by the contractor). The text may be preceded with the statement, "Administrative Information Only -- Not Contractually Binding." This section of the specification shall include information of particular importance to the procuring activity in using this particular specification as a contractual instrument for acquisition of the CPCI either initially or for follow-on procurement.

Background information or rationale which will be of assistance in understanding the specification itself or using the CPCI it specified, may be included herein (e.g., technical data ordering instructions).

Figure IV-11. Software Product Specification Outline (Continued)

10.0 APPENDIX I

This section of the specification shall contain requirements which are contractually a part of the specification but which, for convenience in specification maintenance, are incorporated herein (e.g., requirements of a temporary nature or for limited effectivity). Appendixes may be bound as separate documents for convenience in handling (e.g., when only a few parameters of the program are classified, an appendix containing only the classified material may be established). When parameters are placed in an appendix, the paragraph of Section 10 shall be referenced in the main body of the program specification in the place where the parameter would normally have been specified. Typical data that may be included in computer program development specification appendixes include: (a) Mathematical derivations, (b) Alternate method, (c) Summary of equations, and (d) Definitions of terms.

Figure IV-11. Software Product Specification Outline (Concluded)

Figure IV-12 is the module source listing and figure IV-13 is the module flowchart. Table IV-3 contains a cursory response reasoning (where needed) and response value for each module source listing as applied to this example module.

```
                74/74   OPT=1                FTN 4.6+452      11/03/78  17.20.21      PAGE     1

      1         SUBROUTINE DSRELB(N,R,INA,ALPHA,OUTA)
                C
                C*****************************************************************
                C*                                                               *
                C*TITLE           : RELIABILITY                                  *
                C*                                                               *
                C*MNEMONIC        : RELIAB                                       *
                C*                                                               *
                C*REVISION DATE   : 07/26/78                                     *
                C*                                                               *
                C*PROGRAMMER      : T L PASCHICH                                 *
                C*                                                               *
                C*ABSTRACT        : THIS MODULE CALCULATES RELIABILITY OF A QUESTION.
                C*                  RESPONSES ARE INPUT TO THE MODULE BY ROW AND COLUMN*
                C*                  WHERE COLUMNS CORRESPOND TO EVALUATORS AND ROWS *
                C*                  CORRESPOND TO MODULES. BOTH INPUTS AND OUTPUTS ARE *
                C*                  PARAMETERIZED.                               *
                C*                                                               *
                C*MODULES CALLED  : FPROB                                        *
                C*                                                               *
                C*MODULES CALLING : STAT                                         *
                C*                                                               *
                C*MOD DESCRIPTION : THIS MODULE COMPUTES THREE SETS OF EQUATIONS FROM
                C*                  WHICH THE FINAL RELIABILITY VALUE IS CALCULATED.*
                C*                                                               *
                C*                  1.BASIC EQUATIONS                            *
                C*                    A. V1=SQUARE OF SUM  OF ALL RESPONSES DIVIDED
                C*                          BY THE PRODUCT OF THE NUMBER OF ROWS AND
                C*                          COLUMNS.                             *
                C*                    B. V2=SUM OF SQUARES OF ALL RESPONSES.     *
                C*                    C. V3=SUM ACROSS COLUMNS OF SQUARE OF SUMS ACROSS*
                C*                          ROWS DIVIDED BY NUMBER OF ROWS.      *
                C*                    D. V4=SUM ACROSS ROWS OF SQUARES OF SUMS ACROSS
                C*                          COLUMNS DIVIDED BY NUMBER OF COLUMNS *
                C*                                                               *
                C*                  2. SSQ EQUATIONS                             *
                C*                    A. SSM=V4-V1 (SSQ FOR VARIANCE BETWEEN MODULES)
                C*                    B. SSR=V3-V1 (SSQ FOR VARIANCE BETWEEN RATERS)
                C*                    C. SSWM=V2-V4 (SSQ FOR VARIANCE WITHIN MODULES)
                C*                    D. SSE=(V2/V4)-(V3-V1) (SSQ FOR ERROR VARIANCE
                C*                          AFTER REMOVING VARIANCE BETWEEN RATERS)
                C*                                                               *
                C*                  3. MSQ EQUATIONS                             *
                C*                    A. MSR=SSR/R-1                             *
                C*                    B. MSE=SSE/(R-1)(M-1)                      *
                C*                                                               *
                C*                  REL=1-MSE/MSR                                *
                C*                                                               *
                C*                  THIS MODULE ALSO CALLS THE FPROB MODULE WITH THE
                C*                  ALPHA PARAMETER. FPROB RETURNS F STATISTICS. *
                C*                                                               *
                C*                  THE FOLLOWING DATA ITEMS ARE OUTPUT IN THE OUTPUT
                C*                  PARAMETER ARRAY OUTA :                       *
                C*                   SSM,DFM,MSM,FSM,PROBM                       *
                C*                   SSR,DFR,MSM,FSM,PROBR                       *
                C*                   SSE,DFE,MSE                                 *
                C*                   SST,DFT                                     *
```

Figure IV-12. DSRELB Module Source Listing

```
C*                  REL,REL2                                          *
C*                                                                    *
C*CALLING SEQ      : CALL RELIAB(INA,M,N,ALPHA,OUTA)                 *
C*                                                                    *
C*ARGUMENTS        : INA    INTEGER, ARRAY, INPUT.                   *
C*                          EVALUATON RESPONSES BY RATER AND MODULE.  *
C*                  M       INTEGER, INPUT.                           *
C*                          NUMBER OF ROWS(MODULES) IN ARRAY INA.     *
C*                  N       INTEGER, INPUT.                           *
C*                          NUMBER OF COLUMNS(RATERS) IN ARRAY INA.   *
C*                  ALPHA   REAL, INPUT.                              *
C*                          ALPHA VALUE USED BY ERROR.                *
C*                  OUTA    REAL, ARRAY, OUTPUT.                      *
C*                          OUTPUT DATA ITEMS :                       *
C*                           SSM (SSQ FOR VARIANCE BETWEEN MODULES)   *
C*                           DFM (DEGREES OF FREEDOM FOR MODULES)     *
C*                           MSM (MSQ FOR MODULES)                    *
C*                           FSM (F STATISTIC FOR MODULES)            *
C*                           PBM                                      *
C*                           SSR (SSQ FOR VARIANCE BETWEEN RATERS)    *
C*                           DFR (DEGREES FO FREEDOM OF RATERS)       *
C*                           MSR (MSQ FOR RATERS)                     *
C*                           FSR (F STATISTIC FOR RATERS)             *
C*                           PBR                                      *
C*                           SSE (SSQ FOR ERROR VARIANCE)             *
C*                           DFE (DEGREES OF FREEDOM OF ERROR)        *
C*                           MSE (MSQ ERROR)                          *
C*                           SST (SSQ TOTAL)                          *
C*                           DFT (TOTAL DEGREES OF FREEDOM)           *
C*                           REL (RELIABILITY)                        *
C*                           REL2 (RELIABILITY SQUARED)               *
C*                                                                    *
C*GLOBAL DATA      : NONE                                            *
C*                                                                    *
C*ASSUMPTIONS      : 1. 0<=REL<=1                                    *
C*                    2. IF MSR=0, AN ERROR CONDITION IS FLAGGED BY   *
C*                       ASSIGNING REL EQUAL TO -1.                   *
C*                    3. IF MSE>MSR, AN ERROR CONDITION IS FLAGGED BY *
C*                       ASSIGNING REL EQUAL TO -2.                   *
C*                                                                    *
C*REFERENCES       : THE DERIVATION OF THE MATHEMATICAL MODEL USED BY *
C*                    THIS MODULE IS EXPLAINED IN DETAIL IN THE-------.
C*                                                                    *
C**********************************************************************
C
C
C       ******************************
C       *        REAL VARIABLES      *
C       ******************************
C
      REAL ALPHA,DFE,DFM,DFR,DFT,FSM,FSR,MSE,MSM,MSR,OUTA(17),PBM,PBR,
     *     REL,REL2,SSE,SSM,SSR,SST,SSWM,TEMP,T3,T4,V1,V2,V3,V4
C
C       ******************************
C       *      INTEGER VARIABLES     *
C       ******************************
C
```

Figure IV-12. DSRELB Module Source Listing (Continued)

```
      SUBROUTINE DSRELB     74/74   OPT=1                FTN 4.6+452     11/03/78  17.28.21     PAGE     3

  115                 INTEGER M,R,INA(M,R)
                C
                C       ********************************
                C       *   COMPUTE BASIC EQUATIONS    *
                C       ********************************
                C
  120           C
                      TEMP=0
                      DO 1200 J=1,R
                        DO 1100 I=1,M
                          TEMP=TEMP+INA(I,J)
  125           1100    CONTINUE
                1200  CONTINUE
                      V1=(TEMP*TEMP)/(M*R)
                C
                      TEMP=0
  130                 DO 1400 J=1,R
                        DO 1300 I=1,M
                          TEMP=TEMP+INA(I,J)*INA(I,J)
                1300    CONTINUE
                1400  CONTINUE
  135                 V2=TEMP
                C
                      T3=0
                      DO 1600 J=1,R
                        TEMP=0
  140                   DO 1500 I=1,M
                          TEMP=TEMP+INA(I,J)
                1500    CONTINUE
                        T3=T3+TEMP*TEMP
                1600  CONTINUE
  145                 V3=T3/M
                C
                      T4=0
                      DO 1800 I=1,M
                        TEMP=0
  150                   DO 1700 J=1,R
                          TEMP=TEMP+INA(I,J)
                1700    CONTINUE
                        T4=T4+TEMP*TEMP
                1800  CONTINUE
  155                 V4=T4/R
                      PRINT 111,V1,V2,V3,V4
                  111 FORMAT(1X,4F10.3)
                C
                C       ********************************
  160           C       *    COMPUTE SSQ EQUATIONS     *
                C       ********************************
                C
                      SST=V2-V1
                      SSM=V4-V1
  165                 SSR=V3-V1
                      SSWM=V2-V4
                      SSE=SSWM-SSR
                C
                C       ********************************
  170           C       *     COMPUTE DEGREES OF       *
                C       *  FREEDOM AND MSQ EQUATIONS   *
                C
```

Figure IV-12. DSRELB Module Source Listing (Continued)

```
SUBROUTINE DSRELB     74/74   OPT=1                    FTN 4.6+452        11/03/78  17.28.21       PAGE     4


                C        ********************************
                C
                      DFM=M-1
           175        DFR=R-1
                      DFE=DFM*DFR
                      DFT=M*(R-1)
                C
                      MSM=SSM/DFM
           180        MSR=SSR/DFR
                      MSE=SSE/DFE
                C
                C        ********************************
                C        *       ERROR CONDITIONS       *
           185  C        *IF MSR=0 SET REL & REL2=-1     *
                C        *IF MSE>MSR SET REL & REL2=-2   *
                C        *IF EITHER OF THE ABOVE, SKIP   *
                C        * OVER NORMAL CALCULATION OF    *
                C        * REL & REL2.                   *
           190  C        ********************************
                C
                      IF(MSR.NE.0) GO TO 1900
                      REL=-1
                      REL2=REL
           195        GO TO 2100
                 1900 CONTINUE
                      IF(MSE.LE.MSR) GO TO 2000
                      REL=-2
                      REL2=REL
           200        GO TO 2100
                 2000 CONTINUE
                      REL=1-MSE/MSR
                      REL2=REL*REL
                 2100 CONTINUE
           205  C
                C        ********************************
                C        *IF MSE=0 SET FSM,FSR,PBM, &    *
                C        * PBR=-3, SKIP OVER NORMAL      *
                C        * F STATISTIC CALCULATIONS.     *
           210  C        *OTHERWISE, CALL FPROB TO       *
                C        * CALCULATE F STATISTICS.       *
                C        ********************************
                C
                      IF(MSE.NE.0) GO TO 2200
           215        FSM=-3
                      FSR=FSM
                      PBM=FSM
                      PBR=FSM
                      GO TO 2300
           220   2200 CONTINUE
                      FSM=MSM/MSE
                      CALL DSFPRB(DFM,DFE,FSM,ALPHA,PBM)
                      FSR=MSR/MSE
                      CALL DSFPRB(DFR,DFE,FSR,ALPHA,PBR)
           225   2300 CONTINUE
                C
                C        ********************************
                C        *  STORE OUTPUT DATA IN OUTA   *
```

Figure IV-12. DSRELB Module Source Listing (Continued)

```
      SUBROUTINE DSRELB     74/74   OPT=1                    FTN 4.6+452       11/03/78  17.28.21       PAGE     5

                     C         ********************************
          230        C
                           OUTA(1)=SSM
                           OUTA(2)=DFM
                           OUTA(3)=MSM
                           OUTA(4)=FSM
          235              OUTA(5)=PBM
                           OUTA(6)=SSR
                           OUTA(7)=DFR
                           OUTA(8)=MSR
                           OUTA(9)=FSR
          240              OUTA(10)=PBR
                           OUTA(11)=SSE
                           OUTA(12)=DFE
                           OUTA(13)=MSE
                           OUTA(14)=SST
          245              OUTA(15)=DFT
                           OUTA(16)=HEL
                           OUTA(17)=HEL2
                     C
                           RETURN
          250              END


      SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
    3  DSRELB

VARIABLES     SN  TYPE            RELOCATION
    0  ALPHA     REAL                    F.P.        270  DFE       REAL
  271  DFM       REAL                                272  DFR       REAL
  273  DFT       REAL                                274  FSM       REAL
  275  FSR       REAL                                322  I         INTEGER
    0  INA       INTEGER    ARRAY        F.P.        321  J         INTEGER
    0  M         INTEGER                 F.P.        276  MSE       REAL
  277  MSM       REAL                                300  MSR       REAL
    0  OUTA      REAL       ARRAY        F.P.        301  PBM       REAL
  302  PBR       REAL                                  0  R         INTEGER          F.P.
  303  REL       REAL                                304  REL2      REAL
  305  SSE       REAL                                306  SSM       REAL
  307  SSR       REAL                                310  SST       REAL
  311  SSWM      REAL                                312  TEMP      REAL
  313  TJ        REAL                                314  T4        REAL
  315  V1        REAL                                316  V2        REAL
  317  V3        REAL                                320  V4        REAL

FILE NAMES        MODE
      OUTPUT      FMT

EXTERNALS          TYPE    ARGS
      DSFPXB                  5
```

Figure IV-12. DSRELB Module Source Listing (Continued)

```
      SUBROUTINE DSRELB     74/74   OPT=1                    FTN 4.6+452      11/03/78  17.28.21      PAGE     6

STATEMENT LABELS
  265  111     FMT                        0  1100                        0  1200
    0  1300                               0  1400                        0  1500
    0  1600                               0  1700                        0  1800
  161  1900                             166  2000                      172  2100
  177  2200                             211  2300

LOOPS  LABEL     INDEX     FROM-TO    LENGTH     PROPERTIES
   22  1200    * J         122 126       15B                 NOT INNER
   30  1100      I         123 125        3B      INSTACK
   44  1400    * J         130 134       15B                 NOT INNER
   52  1300      I         131 133        4B      INSTACK
   64  1600    * J         138 144       20B                 NOT INNER
   73  1500      I         140 142        3B      INSTACK
  110  1800    * I         148 154       17B                 NOT INNER
  116  1700      J         150 152        4B      INSTACK

STATISTICS
  PROGRAM LENGTH                 346B      230
        520008 CM USED
```

Figure IV-12. DSRELB Module Source Listing (Concluded)

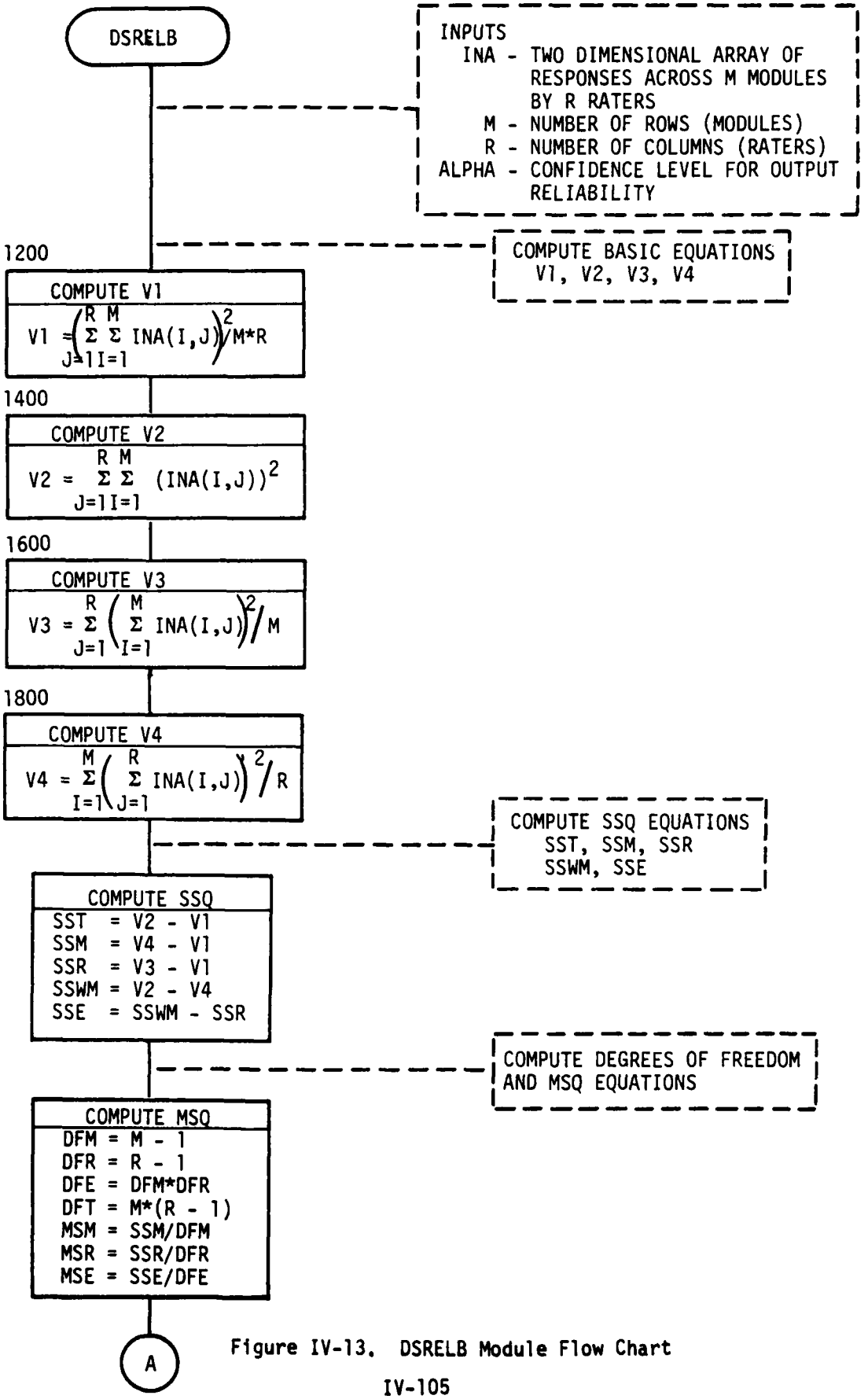


Figure IV-13. DSRELB Module Flow Chart

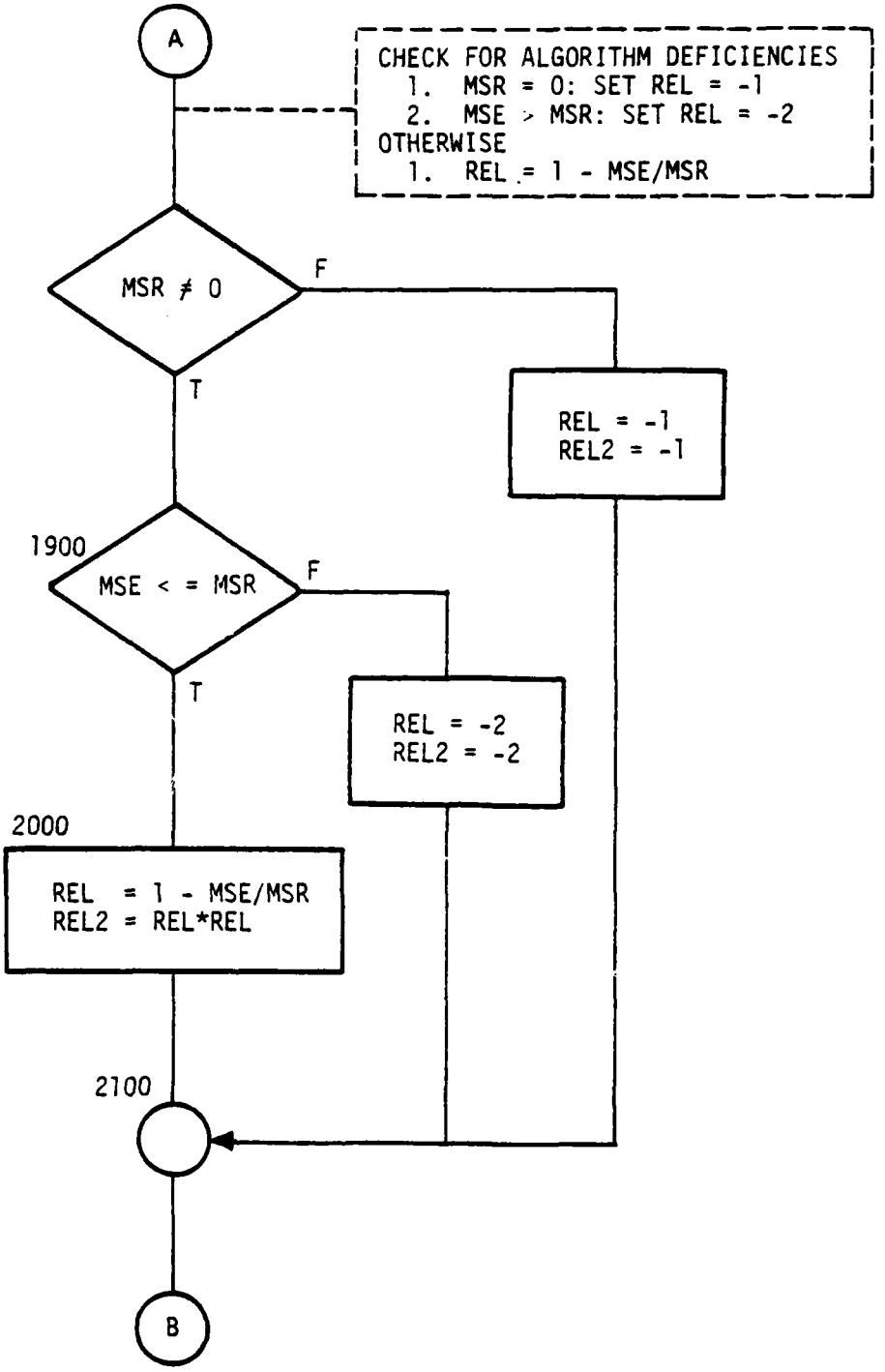


Figure IV-13. DSRELB Module Flow Chart (Continued)

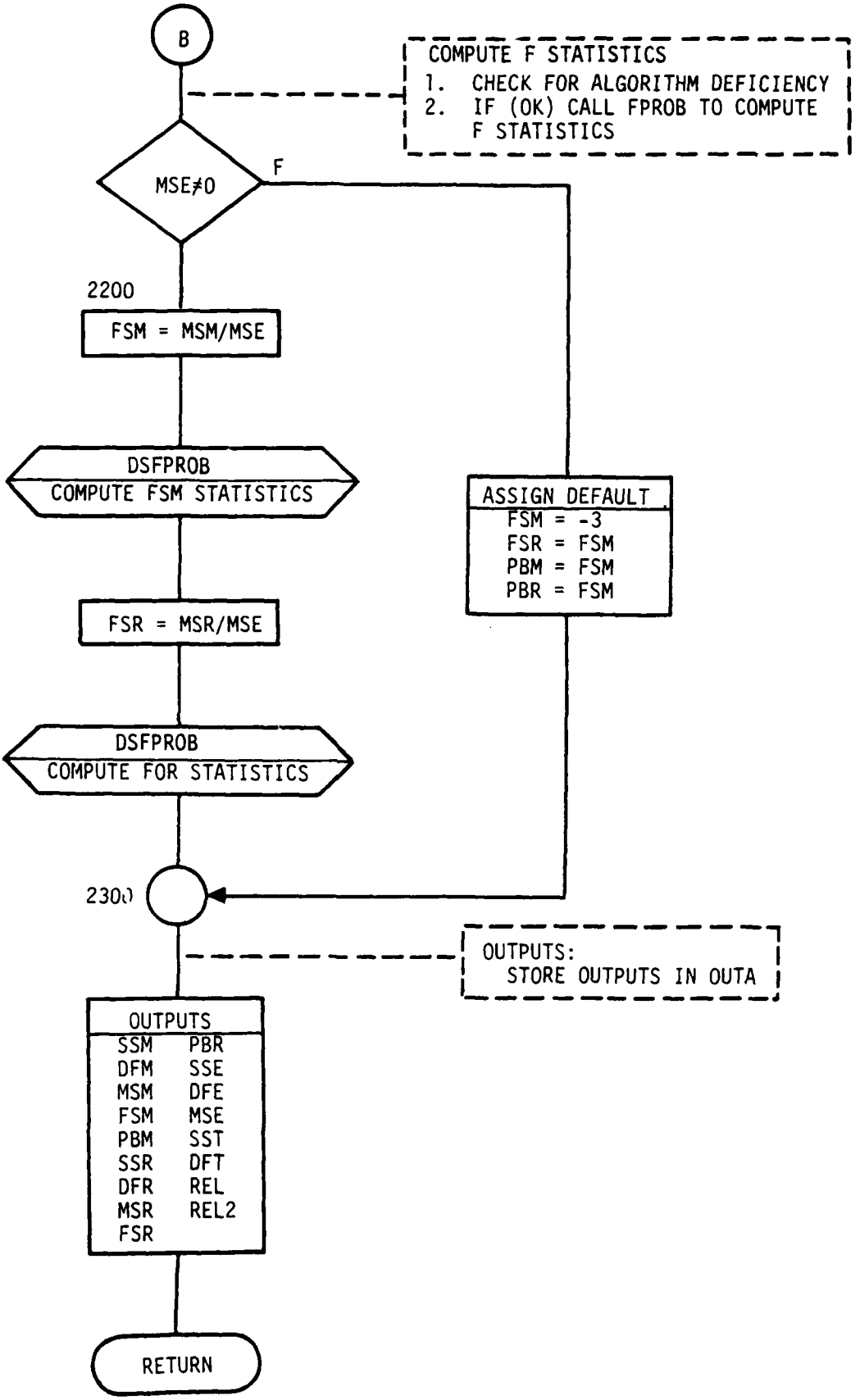


Figure IV-13. DSRELB Module Flow Chart (Concluded)

TABLE IV-3. Module Evaluation Responses

Q#	RESP	DISCUSSION
1	B	The only organization of data is via the arrays INA and OUTA. There appears to be no need to organize other data in any more complex manner. The INA data does appear to be functionally related whereas the OUTA data is slightly less so.
2	A	The control structures used in this module are all of the basic form. The only possible question might be control structures around labels 1900, 2000, 2100 and around labels 2200, 2300. The latter is the classic IFTHENELSE form and the former is an IFTHENELSE form with a nested IFTHENELSE form inside the IF statement part.
3	A	There is no sharing of memory locations.
4	A	There is no global data.
5	A	One entry point.
6	A	One exit point.
7	B	It appears that this module performs the one functional task of computing a reliability number and with some supporting but functionally-related subtasks (F-statistics). It is not certain what purpose some of the data in OUTA (e.g., DFM, SSM) is for since it was not involved in the computation of REL.
8	A	Each subsection of code which represents a functional subtask to the primary task of computing a reliability value is easily recognized.
9	A	The iteration blocks are the DO-loops and each one has a single entry point.
10	A	The iteration blocks are the DO-loops and each one has a single exit point.
11	A	The decision blocks are located from line 192 through line 204, from line 197 through line 204 (nested), and from line 214 through line 225. Each decision block has a single entry point.
12	A	Each of the three decision blocks (as in question 11) has a single exit point. These "points" are, respectively, line 204, line 204, line 225.
13	A	FORTRAN module RETURN automatically causes return to calling module.

TABLE IV-3. Module Evaluation Responses (Continued)

Q#	RESP	DISCUSSION
14	A	There are no variables used for both input and output.
15	B	The inputs are clearly identified. The descriptions are very good but could be better. For example, the functional meaning of the input array could have been better described.
16	B	The outputs are clearly identified. The descriptions are very good but could be better. For example, the functional meaning of sums of squares and reliability and reliability squared could have been better described.
17	C	The purpose is stated but the meaning of the statement is not really that clear (i.e., what is the reliability that is calculated). Because this module computes statistical values whose meaning by name is not obvious, some interpretation would be helpful.
18	A	
19	A	
20	B	There are no limitations as to accuracy, timing, there might be some data I/O problems (see questions 79 and 80).
21	C	The limitations as to the algorithm are identified under the "assumptions" subsection, but a better explanation as to why these are limitations could have been briefly summarized.
22	C	The programmer, module name, and revision data are included. However, for this module it is very necessary that some reference to a more detailed explanation of the mathematical theory is needed. Unfortunately, such a reference is left blank.
23	C	The comments are somewhat useful. Again more information could be included (especially, what the sums of squares represent).
24	B	Very good quantity, just a little more depth would be useful.
25	B	Each transfer of control and destination is not commented, but is clear from the language syntax and the close proximity between transfer and destination. The reason for the transfers is reasonably clear also.
26	A	There do not appear to be any machine dependencies in this module (hence there are no such comments).

TABLE IV-3. Module Evaluation Responses (Continued)

Q#	RESP	DISCUSSION
27	C	There is only one primary function of this module and this function is generally described through some of the comments imbedded within the body of the code.
28	B	Each variable is declared as real or integer (except for the index variables I, J). No other attributes appear to be required (such as units, or other descriptive information).
29	B	Error checks are clearly identified. However, a careful check of the code reveals there probably should be at least one or two other error checks which are not identified. At least there should be an explanation as to why no check is necessary. Checks should be made of the input Rand M to insure that the values are .GT.1. The reason is to prevent a divide by zero (lines 179-181). This lack of a description of why no error check is done only detracts slightly (A to B) on this question (see questions 79 and 80).
30	B	The preamble, data declaration and body seem to have strong formatting conventions which have been followed.
31	B	All variables are declared except for I and J.
32	C	The names are fairly descriptive, but relative to very descriptive names FORTRAN, with its 6-7 character limit, does not allow for much description.
33	C	The DO-loops have been indented while the IF decision blocks have not.
34	B	The ordering of the labels is excellent. The FORTRAN boundaries on the descriptiveness of the labels (numeric) is mildly detracting from "locatability". Some evaluators may still consider the response here to be A rather than B.
35	E	The given cross reference listing is not useful as a cross reference (to the source listing) as to source location of each variable reference. There is some value in having variable information (type) parrotted as a cross check of declaration information. Of course, in this case, a much better cross reference listing could be obtained, but the evaluator should not assume potential but non-available materials could be made available.
36	B	There is a very good correspondence.
37	B	As well as most potential data flow representations.

TABLE IV-3. Module Evaluation Responses (Continued)

Q#	RESP	DISCUSSION
38	B	There is not a one-to-one correspondence, but the lack of correspondence is only a very minor detraction in this case.
39	F	No documentation available (for our example).
40	F	See question 39.
41	F	See question 39.
42	F	See question 39.
43	F	See question 39.
44	B	The comment delineation is very good. This question may deserve an A response.
45	A	Each variable is considered to be of one type.
46	B	The variables primarily do have only one purpose. The only possible exceptions to this are the use of TEMP, and the use of REL, REL2, etc. as a code whenever REL, REL2, etc. can't be calculated because of algorithm deficiency.
47	A	There are no global variables, and clearly local variables have five or fewer characters (as opposed to the maximum number for FORTRAN of 6/7).
48	B	When used (DO-loops, preface block, imbedded comments) there does appear to be uniform indentation.
49	A	There appears to be very strong consistency between the preface block explanations and the source code.
50	A	FORTRAN is an HOL for our purposes.
51	A	The control flow is top to bottom.
52	C	It is not clear whether the computations for SSM, SST, FSM, etc. have a functional purpose. There is some doubt even though they are output.
53	A	There appears to be no such specialized coding techniques.

TABLE IV-3. Module Evaluation Responses (Continued)

Q#	RESP	DISCUSSION
54	B	There does not seem to be any clever programming although it is not absolutely certain since the use of "default" coding of results (lines 193-194, 198-199, 215-218) seems to be clever since it is not explained. Would it not be better to have a simple output code to indicate that all other output is invalid?
55	A	All GOTO-like branches are essential parts of the basic control structures.
56	B	Although there are several labels used, they do appear to be essential.
57	E	Although the summations, divisions, etc. are not difficult, the theory behind the reliability equations does require some higher level mathematical/statistical skills.
58	B	The only compound data structures are INA and OUTA.
59	C	The iterative blocks are all nested as is one of the two decision blocks. However, the nesting level is never more than two deep and the control structures are very simple.
60	A	Standard FORTRAN automatically satisfies this.
61	A	There is no use of BOOLEAN (LOGICAL) expressions except as a control expression for an IF decision and none of these are compound.
62	B	The number of such variables is 19. See figure IV-14.
63	B	The number of unique operators is approximately 18 (exact number depends slightly on the counting algorithm). See figure IV-14.
64	B	The number of unique operands is approximately 52 (exact number depends slightly on the counting algorithm). See figure IV-14.
65	B	The number of executable statements (does not include data statements, comments, CONTINUE, END) is 73.
66	A	There is no I/O.
67	A	There are no apparent machine dependencies.

OPERATORS	OPERANDS		
END OF STATEMENT	ALPHA	SSM	0
DO	DFF	SSR	1
, IN DO	DFM	SST	-1
SUBSCRIPT PARENTHESIS ( )	DFR	SSWM	-2
=	DFT	TEMP	-3
+	FSM	T3	2
*	FSR	T4	3
/	MSE	V1	4
-	MSM	V2	5
.NE.	MSR	V3	6
.LE.	OUTA	V4	7
CALL<RETURN>	PBM	M	8
IF	PBR	R	9
GO TO 1900	REL	INA	10
GO TO 2000	REL2	I	11
GO TO 2100	SSE	J	12
GO TO 2200			13
GO TO 2300			14
			15
			16
			17

CONTROL EXPRESSIONS
J.LT.1 (4)
I.LT.1 (4)
J.GT.R (4)
J.GT.M (4)
MSR.NE. 0
MSE.LE.MSR
MSE.NE. 0

Figure IV-14. Example Operator-Operand-Control Expression Counts

TABLE IV-3. Module Evaluation Responses (Continued)

Q#	RESP	DISCUSSION
68	A	The only repeated parameter is 1 which is used for the loops start value and a few other places. In this case the 1 is best not parameterized.
69	A	There is no such processing dependent code.
70	A	The two dimensions M, R of INA are parameterized.
71	A	There do not appear to be such attributes.
72	B	The module appears to be a non-real-time module although no explicit comments so indicate. It would appear there are no explicit timing requirements or a need for any timing allocation.
73	A	Since the dimensions for the input array INA are parameters this module is not volume limited.
74	B	The functional subparts of this module could be easily replaced (e.g., FPROB, SSM, SSR, ... and other sums of squares) by other algorithmic forms. Also, the whole module could easily be replaced since the only interface is via passed parameters.
75	D	Since the array INA has its dimensions passed as arguments there is no use to check upper bounds. However, it is possible to be passed an M and/or R which could be less than 1. This could cause several repercussions and should be checked. Since the environment for execution has not been defined it is not possible to assume there are any run time options to provide such checks.
76	C	Some of the divide-by-zero checks are providing. However, one outside possibility of a divide-by-zero is when the input M and/or R is either 0 or 1 (see lines 127, 145, 155, 179-181).
77	B	Although the algorithm's theory is fairly complicated, its implementation is fairly straightforward and detailed testing should not be necessary in general.
78	F	No such reference noticed.
79	E	From the comments in question 75 response it is clear that the input data values are not checked. However, most of the values don't have to be checked.

TABLE IV-3. Module Evaluation Responses (Continued)

Q#	RESP	DISCUSSION
80	C	The error codes of -1 and -2 for REL value signal algorithm deficiency. However, there should be some error code for R and/or M .LE.1.
81	C	Although intermediate results pretty obviously can't be selectively collected, there are very few intermediate results (perhas V1, V2, V3, V4) which might be candidates for collection. Many of the intermediate results SSM, DFM, SSR, ... are already output automatically in the output array OUTA.
82	F	There do not appear to be any tracing aids. If the maintenance environment were better known, this response might be different.
83	B	Strong modularity.
84	B	Strong descriptiveness.
85	C	Strong consistency as far as can be determined, but the lack of actual module documentation other than a flowchart drops the response here from a B to a C.
86	B	Strong simplicity.
87	B	For those situations which might warrant change or modification to the source this module is pretty strong.
88	D	The aids are not in general present in the source listings. Perhaps knowledge of the maintenance environment would be beneficial to this rating.
89	B	Overall maintainability gets a solid B.

TABLE IV-3. Module Evaluation Responses (Concluded)

Q#	RESP	DISCUSSION
1-14	5.86	Modularity Average Score
15-35	4.67	Descriptiveness Average Score
36-49	3.78	Consistency Average Score
50-65	5.06	Simplicity Average Score
66-74	5.78	Expandability Average Score
75-82	3.00	Instrumentation Average Score
83-89	4.57	General Questions Average Score
1-82	4.76	Overall Maintainability Score (Equal Weighting/Question)
1-82	4.68	Overall Maintainability Score (Using AFTEC Weight/Test Factor)